Videojocs lliures: Interacció multiplataforma

Antonio Escañuela Rodríguez

Director: Antoni Soto i Riera

Gener de 2010

Índex

1 Introducció **1**

1.1 Objectius i metodologia 1

1.1.1 Plataformes escollides 2

1.1.2 Tasques 2

1.1.3 LaTeX 3

1.2 Estructura del document 4

2 Anàlisi dels dispositius **5**

2.1 Nintendo DS 5

2.1.1 Versions 5

2.1.2 Especificacions tècniques 6

2.1.3 Càrrega de software a Nintendo DS 9

2.2 PlayStation Portable 12

2.2.1 Versions 13

2.2.2 Especificacions tècniques 13

2.2.3 Homebrew 13

3 Anàlisi dels entorns de desenvolupament **15**

3.1 Un toolchain per Nintendo DS: devkitPro 16

3.1.1 La llibreria de sistema: libnds 16

3.2 Un toolchain per PSP: PSPDEV 17

3.3 Scratchbox 18

3.4 Llibreries multi-plataforma 19

3.4.1 Simple DirectMedia Layer 19
3.4.2 libxml2 20
3.5 Eines de compilació 21
3.6 Llibreries per a Nintendo DS 22
3.7 Emuladors de Nintendo DS 24

4 Anàlisi del joc **27**
4.1 Algorisme 28

5 Implementació **31**
5.1 Instal·lació de l'entorn de desenvolupament de Nintendo DS 31
5.1.1 Instal·lació del toolchain devkitPro 32
5.1.2 Instal·lació de la llibreria d'alt nivell PAlib 35
5.1.3 Instal·lació del port de SDL 36
5.2 Compilant llibreries per Nintendo DS 38
5.2.1 Portar SDL_net a Nintendo DS 39
5.2.2 Portant una funció estàndard de C 42
5.2.3 Instal·lació de libxml2 45
5.3 Instal·lació de l'entorn de desenvolupament de PSP 48
5.3.1 Alliberament de la plataforma 48
5.3.2 Instal·lació del toolchain PSPDEV i la llibreria SDL 49
5.3.3 SDL_net i libxml2 50
5.4 Scratchbox2 51
5.5 Adaptació i compilació del joc 55
5.5.1 Compilació condicional 55
5.5.2 Canvis en el joc per a la conversió a Nintendo DS 56
5.5.3 Canvis en el joc per a la conversió a PSP 61
5.5.4 Errors en el codi del 2Pong 65
5.5.5 Compilació 71
5.6 Depuració i proves 74
5.7 Distribució del codi 75

6 Valoració econòmica **76**

7 Conclusions i treball futur **78**

7.1 Conclusions 78

7.2 Possibles millores i ampliacions 80

A Instal·lació i manual del joc **81**

A.1 Instal·lació 81

A.2 Manual del joc 82

B Codi **84**

Agraïments

Primer voldria agrair al nostre director de projecte, en Toni, per guiar-nos en aquest camí, la seva ajuda ha estat de molta utilitat. També voldria agrair a la universitat que m'hagi donat l'oportunitat de conèixer a l'Anna, la meva companya de projecte, ja que és el millor tresor que m'emporto.

Dono gràcies a en Javi per estar sempre al meu costat i fer-me la vida més fàcil. A l'Oscar, l'Iván, l'Eli, en Guiem i l'Adrià per aportar les seves idees i ajuda al projecte. A la Maria, el David i la Haidée pels seus ànims, la seva companyia en la biblioteca i per la seva preocupació per l'estat del projecte. A l'ajut desinteressat de Stephen Stair i altres que hi han col·laborat. Per últim, a la meva mare i germanes per la seva paciència i comprensió. A tots, gràcies.

Capítol 1

Introducció

Gairebé totes les grans empreses d'electrònica de consum ofereixen connectivitat amb o sense fils en els seus aparells. Aquesta connectivitat fa uns anys només la trobàvem en PC's, però cada dia es va estenent a més dispositius, com ara consoles, smartphones i fins i tot targes de memòria i bàscules. Al món dels videojocs aquesta connectivitat ha obert moltes portes al mercat. Ara gairebé tots els jocs inclouen d'alguna manera o altre aquesta opció, ja sigui per jugar en xarxa amb altres persones, mostrar els resultats en un rànquing públic, descarregar actualitzacions, comprar nous complements, etc. Fins i tot han aparegut nous gèneres basats en aquesta característica, com els MMORPG [68], en que el joc es juga exclusivament en xarxa.

Tot i l'estandardització de la pila TCP/IP (nivells 3 i 4 del model OSI) sobre un enllaç físic Wi-Fi (nivell 1 del model OSI), estàndard 802.11 de l'IEEE [57], habitualment es veta la interconnexió entre diferents plataformes per raons de màrqueting. Sabent que l'ús d'aquest protocol és igual per a totes les plataformes que l'usen hem volgut sobrepassar aquesta barrera i demostrar que es pot jugar a un videojoc en xarxa entre dos dispositius de companyies diferents.

1.1 Objectius i metodologia

L'objectiu principal del projecte és portar un videojoc lliure i multi-plataforma ja existent a dispositius de companyies diferents i permetre el joc en xarxa entre ells. Per a assolir-lo usarem només eines de programari lliure [36] i multi-plataforma [18].

Altres objectius són entendre i saber usar correctament les eines de programació multi-plataforma, la metodologia de la compilació creuada, els entorns de desenvolupament del sistemes escollits, i les nocions bàsiques de com es programa i s'estructura un videojoc senzill.

La metodologia emprada en aquest projecte és la cerca d'un videojoc lliure programat en

C++ i SDL originalment per a PC i la migració cap a les plataformes escollides. Primer s'hauran de preparar els entorns de desenvolupament per a cada plataforma, i després fer els canvis que calguin en el codi per adaptar-lo a les particularitats de cada màquina. El procediment per aconseguir aquests objectius es detalla a la secció 1.1.2.

1.1.1 Plataformes escollides

Vam decidir utilitzar la consola portàtil Nintendo DS i el dispositiu mòbil iPhone perquè disposaven de projectes per portar la llibreria SDL a aquestes dues plataformes i totes dues comptaven amb connexió de xarxa sense fils. Aquesta elecció no ens suposava despeses addicionals perquè ja disposàvem dels dispositius.

Altres raons per a l'elecció van ser que la Nintendo DS és la consola portàtil més popular del mercat i el fet de tenir dues pantalles i control tàctil la fa diferent de la resta. En el cas de l'iPhone tenim un dispositiu que no és una consola portàtil però s'ha convertit en una de les plataformes més populars per al desenvolupament de jocs. De les aplicacions disponibles per a iPhone, un 15% són videojocs [10]. La seva pantalla multi-tàctil i l'absència de botons la fa interessant.

Ambdues plataformes són sistemes tancats. En el cas de la Nintendo DS no és possible programar fàcilment donada la dificultat de l'accés al SDK oficial. Pel que fa a l'iPhone sí que podem accedir a un SDK oficial però aquest és tancat i per tant no compleix amb el requisit de treballar amb eines lliures. Aquest fet va suposar un incentiu extra per al nostre projecte.

Encara que no estava previst inicialment una vegada començat el projecte va sorgir la idea de treballar sobre una nova plataforma i vam escollir la PlayStation Portable (PSP), que és una plataforma interessant pel fet de ser la principal competidora de Nintendo DS en el mercat de les videoconsoles portàtils.

Aquest projecte ha estat fruit d'un treball conjunt amb l'Anna Monros en que cada un de nosaltres ha treballat sobre una de les plataformes escollides. Aquesta memòria descriu el treball fet sobre els dispositius Nintendo DS i PSP. La informació sobre el port a iPhone 3G es troba descrita a [119].

1.1.2 Tasques

En aquesta secció dividirem els objectius del PFC en una llista de tasques per esclarir els passos que s'han seguit en la realització del projecte.

1. Estudiar plataformes o llibreries que permetin fer jocs portables.
2. Buscar un joc lliure ja existent, que estigui implementat en llenguatge C/C++ i amb la llibreria SDL, i que tingui joc multi-jugador en xarxa.

3. Escollir les plataformes per portar el joc.
4. Preparar entorns de desenvolupament lliures per tal de poder compilar codi per les plataformes escollides.
   (a) Entendre les nocions de compilació creuada.
   (b) Estudiar l'arquitectura de les plataformes escollides.
   (c) Preparar els dispositius per a que puguin executar codi de tercers.
   (d) Instal·lar un entorn GNU/Linux amb les eines necessàries.
   (e) Buscar i instal·lar entorns de compilació creuada lliures (toolchain) pels dispositius triats.
   (f) Provar que l'entorn ha estat instal·lat correctament.
   (g) Provar altres entorns de desenvolupament.
5. Compilar la llibreria SDL i altres llibreries necessàries per compilar el joc en les plataformes escollides.
6. Portar el joc a les màquines escollides.
   (a) Compilar el joc amb els entorns de compilació creuada.
   (b) Adaptar el codi del joc a les necessitats de cada plataforma.
   (c) Provar i arreglar la comunicació entre els dispositius.
7. Escriure la memòria del projecte.

1.1.3 LaTeX

Per escriure aquesta documentació hem usat LaTeX [60], un sistema de composició de textos utilitzat per a la creació d'articles acadèmics, tesis i llibres tècnics donat la qualitat tipogràfica dels documents realitzats amb ell.

LaTeX és un llenguatge de programació en alt nivell basat en TeX [108]. Donat que les ordres de TeX són a molt baix nivell per a l'usuari és més simple escriure en LaTeX.

A diferència d'altres eines de composició de textos, LaTeX funciona per ordres, pel que vam haver de passar per una fase d'aprenentatge, mitjançant el llibre [120], i altres fonts com [115].

LaTeX és una eina lliure pel que s'adequa perfectament als requisits del nostre projecte.

1.2 Estructura del document

Tal com s'ha esmentat anteriorment aquest projecte ha estat realitzat per dues persones, i per tant, per tenir una visió completa de tota la feina realitzada és necessària la lectura de dues memòries. En les memòries trobarem parts comunes i parts específiques segons la plataforma escollida. Les comunes són:

- Introducció (secció 1)
- Introducció del capítol "Anàlisi dels entorns de desenvolupament" (secció 3)
- Anàlisi de Scratchbox (secció 3.3)
- Llibreries multi-plataforma (secció 3.4)
- Eines de compilació (secció 3.5)
- Anàlisi del joc (secció 4)
- Instal·lació de Scratchbox2 (secció 5.4)
- Errors en el codi del 2Pong (secció 5.5.4)
- Distribució del codi (secció 5.7)

El gruix del document s'estructura en dos grans blocs, anàlisi i implementació. En el primer es fa un estudi dels dispositius escollits analitzant el hardware, després s'estudien els entorns de desenvolupament utilitzats i finalment s'estudia el joc que hem adaptat. En el segon bloc es detallen totes les tasques dutes a terme.

Capítol 2

Anàlisi dels dispositius

En aquest capítol estudiarem les plataformes escollides, per tal de conèixer les màquines en les que anem a treballar i saber quines són les seves capacitats. En ell veurem com carregar programari en aquest tipus de màquines.

2.1 Nintendo DS

La Nintendo DS (amb significat Dual Screen o Developer's System, normalment abreviada com NDS o DS) és una videoconsola portàtil desenvolupada i manufacturada per Nintendo. Va ser llançada primer als Estats Units el novembre de 2004 i més tard a Europa al març del 2005. La seva característica principal és que consta de dues pantalles LCD, essent la inferior una pantalla tàctil. També disposa de micròfon i connexió sense fil, que permet interactuar directament en distancies curtes o bé online a través del servei Nintendo Wi-Fi Connection. Consta de dos slots o ranures, un per cartutxos de Nintendo DS, i un altre per retrocompatibilitat amb cartutxos de Game Boy Advance [38], l'anterior consola portàtil de Nintendo. Poc abans d'entregar aquesta memòria s'ha convertit en la consola de videojocs més venuda de la història amb 145 milions d'unitats [112].

La informació sobre Nintendo DS, les seves versions i les seves especificacions tècniques és una combinació de les fonts [74–76].

2.1.1 Versions

Amb el temps han anat sortint diverses revisions de la consola. En aquesta secció analitzarem en què es diferencien les diferents versions. Podem consultar un resum a la taula 2.1.

Nintendo DS Lite: Llançada a Europa el juny de 2006. Les principals novetats van ser un canvi estètic amb un disseny més estilitzat, un 41% més petit i un 27% menys

pesat que el model original, il·luminació millorada amb la possibilitat d'escollir entre quatre nivells diferents d'il·luminació, i una bateria més duradora.

Nintendo DSi: Llançada a Europa l'abril de 2009. Té un disseny semblant al de DS Lite, però és un model no tan continuista com aquesta, ja que hi ha modificacions al hardware i a les capacitats de la màquina. Inclou dues càmeres de 0.3 megapíxels (una interior i una exterior) que poden ser utilitzades tant per fer fotografies com per utilitzar-les als jocs que ho permetin. Inclou nous LEDs indicadors d'estat, pantalles una mica més grans (0.25" de diferència), una ranura de targes SDHC, nova interfície del menú principal, i nous programes preinstal·lats, com per exemple navegador d'Internet i botiga Nintendo DSi on es poden comprar jocs per descarregar. Com estudiarem al següent apartat d'especificacions tècniques (2.1.2), té un hardware més potent que les seves predecessores, a més s'elimina la retrocompatibilitat amb Game Boy Advance que tenien aquestes.

Nintendo DSi XL (DSi LL al Japó): Va sortir a la venda a Europa el març de 2010. La principal novetat és el grandària de les pantalles, de 4,2" respecte a les 3,25" de la Nintendo DSi i les 3" de la Nintendo DS Lite i la Nintendo DS original. Per tant, les dimensions de la consola també són bastant superiors. Inclou dos stylus (llapis tàctil), un normal i un de la grandària d'un bolígraf. Aquest model no és una successora de Nintendo DSi sinó que un model addicional, pensat per a persones amb menys capacitat visual o problemes per agafar la consola.

Per la realització d'aquest projecte s'ha treballat principalment amb una Nintendo DS Lite, encara que s'intentarà provar el resultat en les versions que sigui possible.

2.1.2 Especificacions tècniques

Necessitem conèixer el hardware en el que anem a programar, per saber quines són les seves possibilitats i peculiaritats a l'hora de programar. No entrarem en molt de detall, perquè no és el nostre objectiu, però es pot obtenir més informació en algunes fonts com [28]. Les dades d'aquesta secció corresponen a la primera versió de la Nintendo DS, però s'ha de tenir en compte que hi ha variacions en el hardware en les diferents versions de la consola. Tenim un resum del hardware i de les diferències entre versions a la taula 2.1.

CPU La Nintendo DS consta de dos processadors: un processador principal ARM946E-S, a 67 MHz i un coprocessador ARM7TDMI a 33 MHz. El propòsit de l'ARM7 és de ser el processador esclau de l'ARM9. L'ARM9 processa els mecanismes de joc i el renderitzat de vídeo, mentre que l'ARM7 processa el só, la Wi-Fi, la pantalla tàctil, els botons i el micròfon. L'ARM7 també pot emular la Game Boy Advance. De fet, l'ARM7 és el mateix processador que el d'aquesta consola, però a 33 MHz en comptes de a 16 MHz. Cal remarcar que per executar un programa calen dos binaris, un per

cada processador. L'ARM9 compta amb una memòria cau de 32kB per instruccions i una de 16 kB per dades.

Memòria La Nintendo DS consta d'una complicada estructura de memòria, que podem veure a la figura 2.1. També tenim un resum de les memòries més destacables a continuació [79]:

Memòria principal És una memòria RAM de 4 MB. Generalment guarda l'executable de l'ARM9, així com la gran majoria de les dades del joc. És compartida pels dos processadors i poden accedir-hi alhora. En cas de conflicte, el processador ARM7 té prioritat, però això es pot canviar.

IWRAM És una memòria RAM ràpida de 64 kB exclusiva pel processador ARM7, que la utilitza com a memòria pseudo-cau per guardar instruccions i dades, per així minimitzar l'accés a memòria principal. Per tant és necessari que els executables per aquest processador siguin petits per tal que càpiguen totalment en la IWRAM.

RAM compartida Hi ha dos blocs de 16 kB de memòria RAM de 32 bits ràpida que poden ser assignats als dos processadors. Com que tots dos processadors hi poden accedir, aquests blocs de memòria s'utilitzen perquè es puguin comunicar. Les dues CPUs no poden accedir al mateix bloc a la vegada.

Video RAM La Nintendo DS té nou bancs de memòria de video (un total de 656 kB) que poden tenir molts usos diferents. Poden guardar sprites, textures, tiles, o un mapa de pixels per pintar a pantalla. És una memòria flexible però limitada.

GPU Té dos motors de renderització 2D (un per cada pantalla), que són variants millorades del controlador de vídeo de Game Boy Advance. També té un motor de renderització 3D, que pot ser assignat a una de les dues pantalles. Està limitat a 6144 vèrtexs per frame (uns 2048 triangles), que pot semblar poc, però s'ha de tenir en compte el tamany de la pantalla de la consola. Es podria resumir en que té un avançat sistema de representació 2D, però un modest sistema 3D. La GPU és un xip propi, integrat al mateix xip que els dos ARM.

Àudio Pel que es refereix a só, la Nintendo DS disposa de 16 canals d'àudio independents, que poden reproduir música en 8 bits, 16 bits o format ADPCM. Cada canal té la seva pròpia freqüència, volum i velocitat. Consta de dos altaveus, per tant pot reproduir en estèreo.

Wifi Disposa de xarxa sense fil integrada, capaç d'oferir connectivitat estàndard Wi-Fi 802.11b, i també connectivitat via un protocol propietari de Nintendo anomenat NiFi (per Nintendo Wi-Fi). La Wi-Fi s'utilitza per comunicacions a través d'Internet i el protocol NiFi entre consoles Nintendo DS (mode *ad hoc*). El processador ARM7 és l'únic que pot accedir-hi.

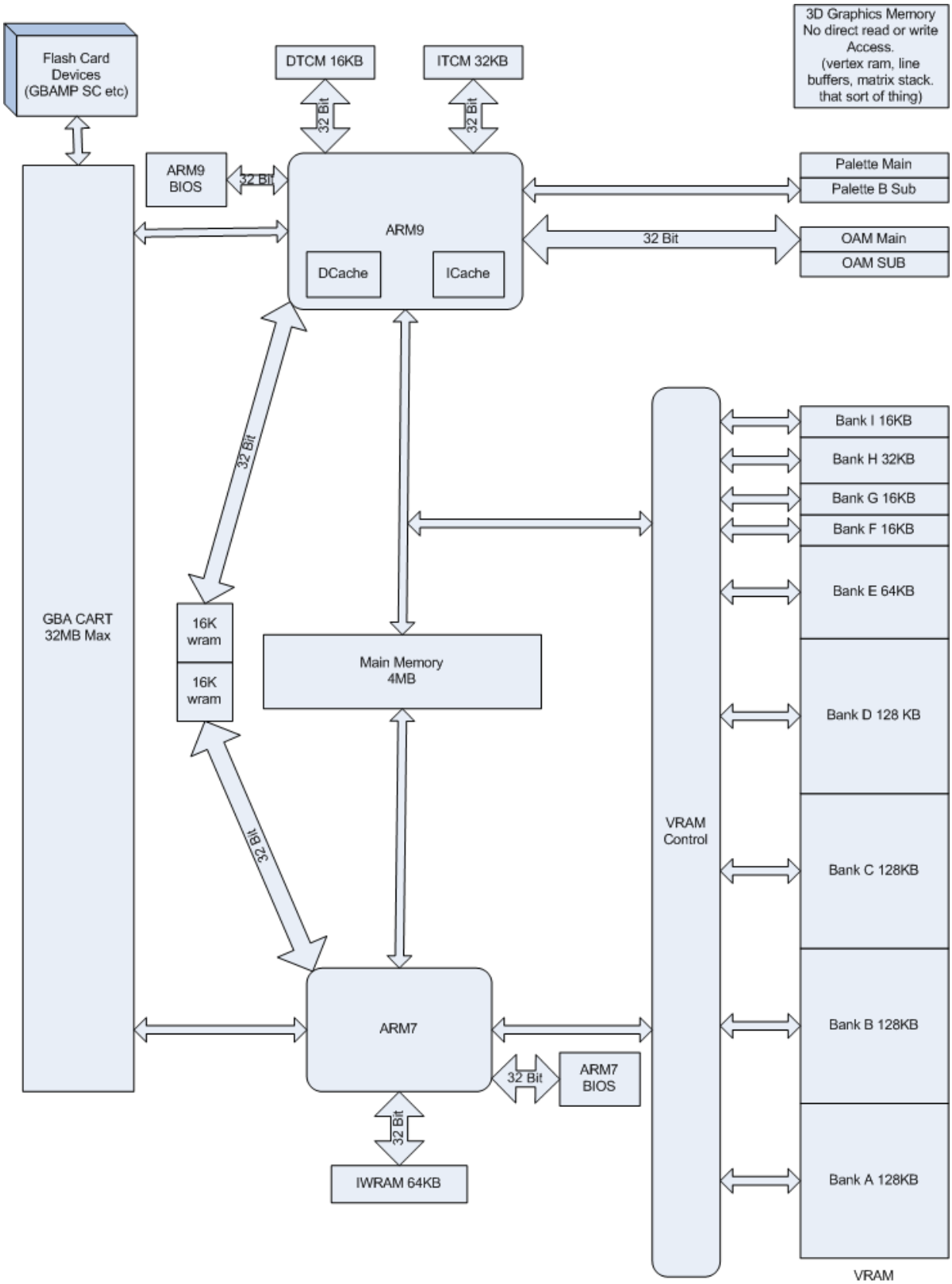

Figura 2.1: Mapa de memoria de Nintendo DS [79].

Input Una de les característiques més rellevants de la Nintendo DS és la seva innovació en el control. Disposa d'un controlador de 12 botons. Deu d'aquests botons són heretats de la Game Boy Advance (Up, Down, Left, Right, A, B, L, R, Start, Select), als que se li afegeixen X i Y. Els deu botons de la GBA són accessibles des dels dos processadors, però els dos botons nous només es poden accedir des del processador ARM7. També disposa d'una pantalla tàctil, la qual treballa mitjançant una interfície sèrie només accessible des de l'ARM7. La pantalla és un model TSC2046 de Texas Instruments. Per últim, consta d'un micròfon, que també ha de ser accedit des del processador esclau.

	Nintendo DS	Nintendo DS Lite	Nintendo DSi	Nintendo DSi XL
Pes	300 grams	218 grams	214 grams	314 grams
Dimensions	148.7 x 84.7 x 28.9 mm	133 x 73.9 x 21.87 mm	137 x 74.9 x 18.9 mm	161 x 91.4 x 21.2 mm
Mida pantalles	3.0"	3.1"	3.3"	4.2"
Resolució pantalles	256x192	256x192	256x192	256x192
Duració bateria	De 10 a 14 hores (850 mAh)	De 15 a 19 hores (1000 mAh)	De 9 a 14 hores (840 mAh)	De 13 a 17 hores (1050 mAh)
Nivells de lluminositat	1 nivell (on/off)	4 nivells	5 nivells	5 nivells
Stylus (llapis tàctil)	75 x 4 mm	87.5 x 4.9 mm	92 x 4.9 mm	129.3 x 10 mm
Ranures de targeta	DS, Game Boy Advance	DS, Game Boy Advance	DSi/DS, SD	DSi/DS, SD
Emmagatzemament	256 kB de memòria flash sèrie	256 kB de memòria flash sèrie	256 MB de memòria flash interna	256 MB de memòria flash interna
RAM	4 MB de RAM mòbil, expansible per la ranura de Game Boy Advance	4 MB de RAM mòbil, expansible per la ranura de Game Boy Advance	16MB	16 MB
CPU	ARM9 a 67 MHz i ARM7 a 33 MHz	ARM9 a 67 MHz i ARM7 a 33 MHz	ARM9 a 133 MHz i ARM7 a 33 MHz	ARM9 a 133 MHz i ARM7 a 33 MHz
Càmera	No	No	Dues càmeres (una exterior i una interior) de 640x480 (0.3 megapíxel)	Dues càmeres (una exterior i una interior) de 640x480 (0.3 megapíxel)

Taula 2.1: Hardware de les diferents versions de Nintendo DS

2.1.3 Càrrega de software a Nintendo DS

Es denomina *homebrew* (programari casolà no oficial) a les aplicacions i jocs creats per programadors per a qualsevol plataforma, però generalment consoles de videojocs propietàries, sense cap tipus de codi o maquinària cedida per les empreses distribuïdores de consoles. En altres paraules, programes per a plataformes de videojocs que no són típicament programables per usuaris [50].

La Nintendo DS consta d'un kit de desenvolupament oficial però que no és fàcil d'aconseguir, s'ha de demostrar el teu mèrit d'obtenir la llicència mostrant un excel·lent joc o una part de software molt novedosa. A més has de demostrar una història financera impecable i uns alts ingressos durant el temps que la llicència de Nintendo sigui vàlida. A més la inversió per poder adquirir l'equipament oficial és molt alta. Tot això fa que una persona

normal no pugui produir jocs propis. No hi ha informació de com obtenir aquest kit de desenvolupament oficial ni tan sols a la seva pàgina oficial [81], els hi has de demanar la informació explícitament.

El tema de la legalitat del homebrew és delicat. Si es necessita modificar el hardware per poder executar el homebrew, llavors és il·legal, perquè els sistemes estan protegits per lleis de copyright i acords amb l'usuari final. Però no és el cas de la Nintendo DS, que no s'ha de modificar per poder executar homebrew, sinó que només s'ha d'utilitzar una targeta especial per carregar jocs i una memòria MicroSD. Tot i això, la legalitat d'aquests aparells sovint es posa en dubte, degut a que també serveixen per carregar còpies de jocs originals, i aquest ús fins i tot ha fet que recentment es prohibeixin en alguns països com el Regne Unit [80]. Però cal aclarir que es declaren il·legals per frenar l'ús que se'ls hi dona per a la pirateria, però la targeta en sí és completament legal, només es tracta d'un mètode per accedir a emmagatzemament extern, que d'altre manera no es podria fer. Per tant, són totalment legals si s'utilitzen per altres propòsits, com carregar còpies dels teus propis jocs originals, o per carregar codi propi. Descarregar i utilitzar un joc homebrew distribuït gratuïtament tampoc infringeix cap llei [27].

Mètodes de càrrega de Homebrew

Com hem vist, la Nintendo DS no compta amb un *sistema d'emmagatzematge*, i per tant per carregar homebrew necessitarem d'un maquinari especial. Existeixen molts tipus de dispositius diferents, aquí explicarem els mètodes principals. Aquesta informació ha estat extreta de [35, 51, 54].

Un *flashcart* és un cartutx dissenyat per connectar-se a una de les ranures de la Nintendo DS (ja sigui la ranura de jocs de Nintendo DS o la ranura de Game Boy Advance), la funció del qual és permetre accedir a una memòria reescribible, per poder carregar binaris (ja siguin còpies de seguretat o homebrew), entre d'altres funcions. El nom de flashcart és una abreviació de Flash Cartridge, encara que també és habitual trobar-ho referenciat com Flash Card. No hi ha molt consens en la seva denominació.

Els primers flascarts anaven a la ranura de cartutxos de GBA (Game Boy Advance, també anomenada "Slot 2") i es combinaven amb un dispositiu carregador a la ranura de cartutxos de Nintendo DS (anomenada "Slot 1"). A aquesta combinació de dues eines es sol anomenar de "Slot 2" o de "Primera Generació". Més tard, es van desenvolupar targetes que només utilitzaven la ranura de Nintendo DS (Slot 1) per emmagatzemar i carregar programari homebrew. Aquests mecanismes es solen anomenar de "Segona Generació" o simplement de "Slot 1".

Hi ha algunes diferències entre els dispositius de Slot 1 i de Slot 2. Com hem vist, els de Slot 2 necessiten un mètode de càrrega a banda del flaschart. Els de Slot 2 no funcionen a partir de Nintendo DSi, donat que no tenen ranura de Game Boy Advance. Els de Slot 1 no poden carregar jocs o homebrew de Game Boy Advance.

Els diferents models flashcarts disponibles es diferencien en mida, compatibilitat amb les ROMs comercials, característiques especials incloses (com per exemple reproductors multimèdia), disponibilitat i preu. Per emmagatzemar el homebrew, els flashcarts utilitzen o bé memòria flash interna o bé memòries flash externes com MicroSD o CompactFlash.

No farem un estudi de tots els models que existeixen, però nombrarem uns quants a mode d'exemple. De Slot 2 tenim tots els flashcart de Game Boy Advance, el SuperCard, el Max Media Dock i el M3 Perfect. De Slot 1 tenim el DSTT, R4DS, SuperCard DS One, M3 Real, per nombrar-ne només alguns exemples famosos.

Com hem vist, per executar homebrew en una Nintendo DS utilitzant un flashcart de Slot 2 es requereix una *eina d'inicialització* (booting tool), que és un dispositiu que envia el punter d'instrucció de la consola a l'adreça correcta, permetent als programes executar des del nostre emmagatzemament. No es requereix aquesta eina d'inicialització per als dispositius de Slot 1. A continuació indiquem algunes eines d'inicialització que existeixen:

PassMe És un dispositiu que connectarem a la consola pel port de targetes Nintendo DS (Slot-1). En un funcionament normal, el cartutx original de Nintendo DS és l'encarregat de comunicar a la consola la adreça on s'emmagatzema el començament del codi, i llavors la consola salta a aquesta posició i l'executa. El PassMe és un dispositiu que es situa entre la consola i un cartutx original, i la idea és que que bàsicament deixa passar (d'aquí el nom passthrough que també se li atribueix) totes les comunicacions entre la consola i el cartutx fins que el punter d'execució és enviat a la Nintendo DS. Llavors aquest punter és reemplaçat per l'adreça de memòria que es vol de l'Slot-2, de manera que la Nintendo DS comença l'execució des del nostre emmagatzemament extern en comptes del cartutx de Nintendo DS. Aquest sistema només funciona amb les primeres versions del *firmware* [1] de Nintendo DS.

WifiMe La Nintendo DS té la possibilitat de descarregar demos usant les capacitats WiFi integrades. La idea és emular una estació de descàrrega de Nintendo DS utilitzant un adaptador especial de WLAN, un driver personalitzat i una aplicació anomenada WMB (Wireless Multiboot). Usant això podrem enviar demos a la Nintendo DS de manera que, semblant al que fa el PassMe, es pot enviar un binari modificat per tal que el punter a l'inici del codi apunti a l'adreça que volem de l'emmagatzemament de GBA. Els principals inconvenients d'aquest mètode són que haurem de tenir un PC a prop i que haurem de modificar el controlador de la targeta WiFi del PC (pel que només la podrem usar per fer WifiMe fins que es restaurin els controladors del fabricant). A més, només funciona amb les primeres versions del firmware de Nintendo DS [113].

FlashMe Es tracta d'un firmware modificat, que s'instal·la sobre el firmware original i permet executar directament les aplicacions homebrew a l'Slot-2 sense necessitat d'altres

[1] El firmware és un programa integrat en l'electrònica d'un dispositiu que estableix la lògica de més baix nivell que controla els circuits electrònics del dispositiu [69].

aparells. També es poden enviar aplicacions via WMB. El seu avantatge principal és que no es necessita cap hardware addicional per carregar homebrew. Però com que és un programa homebrew, per instal·lar-ho si que farà falta que disposem de la possibilitat prèvia d'executar homebrew, és a dir, per instal·lar-ho si que necessitarem PassMe o WifiMe. A més, FlashMe anula la garantia, i encara que el procés no és molt complicat, és arriscat donat que si no es fa bé es pot deixar inservible la consola.

NoPass Més endavant es va descobrir com funcionava l'encriptació del cartutx de Nintendo DS, i això va permetre una alternativa al PassMe, anomenada NoPass, que és un cartutx de Nintendo DS que diu a la consola d'executar codi de la ranura de GBA. És la mateixa idea i procediment que el PassMe, però no sobresurt de la consola, no necessita un cartutx original de Nintendo DS i és compatible amb totes les versions del firmware (fins a Nintendo DS Lite). El primer NoPass va ser el Max Media Launcher de Datel.

Maquinari triat

Encara que existeixen emuladors de Nintendo DS (que estudiarem més endavant), és interessant fer proves amb el hardware final. Per tant, necessitarem algun mètode de càrrega de homebrew. En aquest projecte s'ha treballat principalment amb un "M3 DS Real" [66], de Slot 1, encara que també s'han fet proves amb un "SuperCard Lite" [107] de Slot 2 (amb el NoPass SuperKey).

2.2 PlayStation Portable

La PlayStation Portable (a partir d'ara PSP) és una videoconsola portàtil de Sony Computer Entertainment. Tercera consola de la línia PlayStation i primera incursió de Sony al mercat de les portàtils, fou presentada oficialment el 13 de maig de 2003. Fou comercialitzada el 12 de desembre de 2004 al Japó, el 24 de març de 2005 a Amèrica del Nord i l'1 de setembre del mateix any a Europa [91].

A més de l'ús de videojocs, la PSP és capaç de reproduir vídeo, fotografies i música. Aquest dispositiu es basa en l'Universal Media Disc (UMD), format propietat de Sony. L'UMD és un disc òptic de doble cara i 60 mm que pot emmagatzemar fins a un total d'1,8 GB de contingut dins d'una cuirassa de plàstic especialment dissenyada per la companyia. A més de l'UMD, la PSP empra unes targetes Memory Stick Duo per desar les dades de les partides, fotografies, música i video. Altres característiques són una gran pantalla panoràmica, i connectivitat amb PlayStation 2, PlayStation 3 i altres PSP.

La PSP té compatibilitat Wi-Fi, que s'utilitza tant per jocs multijugador local o a través d'Internet, com per actualitzar jocs, descarregar continguts addicionals o inclús navegar directament per la xarxa gràcies al navegador inclòs a la versió 2.0 del seu firmware.

Un motiu que la feia una plataforma atractiva per treballar-hi és que sigui la principal competidora de Nintendo DS al mercat de les consoles portàtils, encara que amb el temps s'han anat orientat cap a mercats diferents: la Nintendo DS cap a un públic més casual i infantil, i treu més profit dels jocs 2D i la pantalla tàctil; mentre que la PSP s'ha orientat cap a un públic més habitual i treu més rendiment del 3D i de l'oci multimèdia.

2.2.1 Versions

Hi ha diverses versions i models de PSP. L'original és la PSP-1000 (també coneguda col·loquialment com el model Fat per ser la més pesada de totes). Més endavant van sortir noves versions: la PSP-2000 i la PSP-3000 (també conegudes col·loquialment com els models Slim), models més lleugers i amb alguns canvis en el hardware, com per exemple més memòria RAM o millores a la pantalla. Més endavant va sortir també un nou model, anomenat PSP Go, orientat al mercat de les descàrregues digitals, ja que no té suport per discs UMD i en el seu lloc compta amb una memòria flash interna de 16 GB on guardar els jocs que s'han de descarregar de la PlayStation Store (botiga oficial) [93]. Aquesta última també consta d'importants canvis estètics i de hardware.

2.2.2 Especificacions tècniques

Per no copiar la taula d'especificacions tècniques, podem consultar-la a [92].

2.2.3 Homebrew

El *firmware* és un programa integrat en l'electrònica d'un dispositiu que estableix la lògica de més baix nivell que controla els circuits electrònics del dispositiu [69].

El primer firmware de la consola va ser el 1.00, però només va ser distribuït al Japó. Es va trobar un *exploit* [2] per carregar homebrew, que seria arreglat per Sony amb el firmware següent. L'accés al kernel era bastant complet, el que permetia als hackers treballar amb més facilitat sobre aquesta versió. El 1.50 és el firmware natiu de les PSP americanes i europees. És famós per tenir l'exploit que permet carregar homebrew. Fins a l'aparició del primer custom firmware (el 2.71 SE-A) seria l'única versió en la que es podia córrer homebrew. Més tard apareixerien els firmwares oficials 1.51 i 1.52, en els que es corregia les falles de seguretat que permetien carregar homebrew des de la versió 1.50 [52].

A partir d'aquest moment, Sony cada cop més va anar invertint en seguretat i tancant aquests forats de seguretat mitjançant nous models de hardware i nous firmware, i cada

[2] Un programa amb l'objectiu d'aprofitar un error, amb el fi de violar la seguretat per accedir a un sistema de forma no autoritzada.

cop era més difícil de saltar la seguretat. A partir de les PSP-3000 amb el firmware 6.20 no han aconseguit alliberar-se.

Degut a la gran varietat de firmwares i versions de la consola, molts homebrews funcionen en una videoconsola sí i en una altra no. Alguns backups s'executen en mode "M33" i d'altres en mode "Sony 9660". Els backups carregaran si són compatibles amb el firmware en qüestió, si hem seleccionat adequadament els paràmetres del programa i si hem posat els seus arxius al directori adequat (GAME, ISO, etc.) segons el firmware personalitzat que operi en la consola.

Els *Custom Firmware* (firmware personalitzats, CF o CFW) són firmwares alternatius, no oficials, que ens permeten executar homebrew. Van ser iniciats i son creats en la seva majoria per el Team M33, del que forma part el programador conegut pel sobrenom de Dark Alex.

Modificar el firmware de la nostra màquina no és il·legal, però ens permet usar còpies de videojocs originals i això sí que és il·legal. Quan canviem el firmware potser estem trencant el contracte amb Sony, però com a molt perdrem la garantia de la nostra màquina.

Capítol 3

Anàlisi dels entorns de desenvolupament

Un entorn de desenvolupament és el conjunt de processos i eines de programació usades per crear un programa o producte de software. Pot implicar també un entorn físic. En eines de programació s'inclouen eines com compiladors, llibreries i altres eines per a l'ajuda de la programació, com repositoris de codi, eines de disseny de software, eines per a la documentació, simuladors, etc.

Podem dir que la preparació d'aquests entorns no sempre és trivial. Segons la plataforma podem trobar entorns ja muntats i provats i que es distribueixen com a entorns funcionals, que es solen anomenar *toolchain*.

Un toolchain (que podríem traduir com cadena d'eines) és un conjunt de programes informàtics utilitzats per a crear un determinat producte (normalment un altre programa o sistema informàtic). Els diferents programes se solen utilitzar en cadena, de manera que la sortida de cada eina sigui l'entrada de la següent, encara que actualment es fa un abús del terme per referir-se a qualsevol tipus d'eines de desenvolupament enllaçades [109].

Per a la realització d'aquest projecte s'ha usat un toolchain per Nintendo DS, que s'analitza a la secció 3.1, un altre per PSP a la secció 3.2, i s'ha experimentat amb un altre tipus d'entorn de desenvolupament, que s'explica a la secció 3.3.

Els nostres entorns de desenvolupament compten amb un compilador i les llibreries de sistema 3.1.1 i 3.6. També necessitarem altres llibreries que són usades pel joc, i que explicarem a 3.4.

Donat que les llibreries i el joc que compilarem estan escrites en llenguatge C/C++ necessitem un compilador d'aquest llenguatge que generi executables per al nostre dispositiu. En el nostre projecte usem el GCC [39], de les sigles de GNU Compiler Collection, perquè treballem sobre GNU/Linux.

Compilar en C/C++ des d'una plataforma com Nintendo DS o PSP no és possible ja

que es tracta de dispositius encastats i sense sistema operatiu. Un cop s'ha descartat la possibilitat de compilar des del mateix dispositiu, ens cal estudiar el concepte de compilació creuada.

La *compilació creuada* consisteix en crear codi executable per a una plataforma diferent d'aquella en la que es compila [17]. Això és necessari quan volem executar codi en una plataforma en la que no el podem compilar (degut a les seves limitacions) o no és còmode fer-ho, com és el nostre cas. És a dir, compilarem els binaris en un PC però segons l'arquitectura d'una Nintendo DS o una PSP, de manera que finalment puguin ser executats en aquestes últimes.

3.1 Un toolchain per Nintendo DS: devkitPro

La informació sobre el toolchain de Nintendo DS i sobre compilació en Nintendo DS principalment ha estat extreta de [4, 58].

Ja hem vist que per desenvolupar per Nintendo DS les eines per desenvolupadors comercials no estan disponibles per desenvolupadors particulars, però per sort existeix un toolchain alternatiu, el nom del qual és *devkitPro* [23].

Aquest paquet dóna suport a més consoles a banda de la Nintendo DS (GameBoy Advance, GP32, PlayStation Portable, GameCube, etc.). Com hem vist, la Nintendo DS té una arquitectura ARM, per tant necessitarem en concret l'eina *devkitARM* inclosa en el devkitPro i que ens prové de tot el que necessitem. Per exemple inclou el compilador gcc ja mencionat. En concret la versió devkitarm_r26 que jo vaig instal·lar primer portava la versió 4.4.0 del compilador. També inclou l'enllaçador ld, o l'eina ndstool per crear els executables en format `.nds`.

Amb devkitARM és possible escriure programes amb totes les funcions del nucli de C, a més de les funcions estàndard que aporta el devkitARM (abs, atoi, free, etc.). Però per sortida de text, gràfics, entrada per botons o per pantalla tàctil, necessitarem alguna cosa més. Es podrien realitzar totes aquestes operacions en el nivell més baix. Per exemple, escrivint a un registre de 32 bits que hi ha a l'adreça 0x4000000 per controlar la pantalla. Però per facilitar aquesta tasca està la llibreria *libnds*.

3.1.1 La llibreria de sistema: libnds

La llibreria *libnds* [62] (abans anomenada NDSLIB) dóna suport a la programació de totes les característiques de la Nintendo DS, incloent: pantalla tàctil, hardware 2D/3D, entrada per botons, micròfon, etc. És una llibreria de baix nivell que fa una petita abstracció del hardware, gràcies a unes primitives que oculten detalls tècnics no desitjats, com ara la ubicació real dels registres en la memòria.

libnds va ser creada per joat (Michael Noland) i dovoto (Jason Rogers), i ara és mantinguda per WinterMute (Dave Murphy). Avui dia libnds és una llibreria molt útil que es calcula que és usada per un 96% de la comunitat de programadors homebrew [58], donat que les llibreries de més alt nivell també es basen en libnds.

A més de les constants numèriques, libnds defineix diverses funcionalitats útils:

- Gestió del sistema d'arxius
- L'accés a les funcions del BIOS
- Operacions de lectura / escriptura de la targeta
- Gestió DMA
- Interrupcions
- IPC (InterProcess Communication)
- Gestió de l'entrada
- Gestió del temps
- Video
- Àudio
- Temporitzadors
- Sortida de la consola
- Diversos

3.2 Un toolchain per PSP: PSPDEV

Existeixen diversos entorns de desenvolupament per PSP: el devkitPSP [25] del paquet devkitPro, i el que s'ha utilitzat en aquest projecte, el PSPDEV [98]. Aquest està basat en GCC 4.0.1, Binutils 2.16 i Newlib 1.13.0.

El toolchain PSPDEV es troba dins un paquet anomenat PSPSDK (Software Development Kit) [99]. Encara que els dos termes fan referència al concepte "entorn de desenvolupament" són dos conceptes diferents: el PSPDEV és el toolchain en el sentit estricte de combinació d'eines per a la compilació, i el PSPSDK és un paquet que a més del toolchain PSPDEV inclou més eines, incloses les llibreries SDL i SDL_mixer. PSPSDK es distribueix sota llicència BSD de tres clàusules [65].

Ens vam decidir per PSPDEV en comptes de devkitPSP perquè en el moment de realitzar l'estudi de la plataforma i dels entorns de desenvolupament ens vam trobar que PSPDEV estava més recolzat per la comunitat: la majoria de les fonts que consultàvem parlaven d'aquest toolchain com per exemple el tutorial [96] o la comunitat de desenvolupadors [97].

3.3 Scratchbox

La compilació creuada pot comportar problemes, per exemple, a l'hora d'usar llibreries que estan presents a ambdós sistemes (on es compila i on s'executa el codi), pot ser que perdem el control i no sapiguem quina versió de la llibreria estem usant. Per això ens vam plantejar l'ús d'una eina que ens permetés evitar aquests problemes. El nostre director de PFC ens va recomanar l'estudi d'Scratchbox [102] donat que coneixia el seu ús per al desenvolupament per a Maemo [67], el sistema operatiu d'alguns dispositius Nokia [85], com el 770, N800, N810 i N900.

Scratchbox crea un entorn segur de compilació creuada sobre GNU/Linux basat en la idea de chroot [14]. Chroot en un sistema operatiu Unix canvia el directori arrel a un de diferent de "/". Un programa que s'executa en un entorn chroot no pot accedir a arxius fora d'aquest directori creant una mena de gàbia.

Suporta arquitectura ARM i x86, cosa que el fa idoni per al nostre cas, ja que l'arquitectura de la CPU de l'iPhone i la Nintendo DS és ARM. Encara que es pot instal·lar sobre les dues arquitectures és més habitual usar-lo per fer desenvolupament per a ARM sobre una màquina x86. Scratchbox usa Qemu [100] per emular un processador ARM. Està llicenciat sota LGPL i per tant compleix amb les premisses del PFC.

Scratchbox ofereix diversos toolchains per ser usats amb ell i també ofereix l'opció d'usar-ne un de propi, com és el nostre cas. Durant la instal·lació vam tenir problemes en aquest punt, ja que no sabíem com incloure'l, per això vam demanar informació a la llista de correu oficial. Un dels usuaris ens va recomanar Scratchbox2, un projecte separat de l'Scratchbox original. Donada la seva senzilla instal·lació ens vam decantar per aquesta opció.

Cal dir que Scratchbox2 disposa de molt poca documentació i no s'especifica amb claredat les diferències que presenta amb Scratchbox. El projecte va ser abandonat pel seu creador (Lauri Leukkunen) durant la realització d'aquest PFC, al març del 2010. Segons s'informa a la seva web ara el projecte està mantingut per Nokia.

Finalment no s'ha usat Scratchbox per a la realització del PFC per un tema de comoditat, ja que amb els makefiles creats per nosaltres teníem un major control. Encara que l'hem instal·lat i hem fet proves sobre ell que es detallen a la secció 5.4.

3.4 Llibreries multi-plataforma

L'objectiu d'aquest projecte és portar un joc multi-plataforma, per tant es buscarà un joc que estigui implementat d'una manera fàcilment portable i que usi llibreries també portables i preferiblement disponibles a la nostra plataforma. A l'hora de buscar el joc ens hem centrat en que estigués implementat amb una llibreria gràfica que no sigui depenent del hardware. Existeixen llibreries que permeten tractar el hardware de forma transparent, de la mateixa manera sigui quina sigui la màquina en la que es treballa. Les eines lliures multi-plataforma més conegudes són les Simple DirectMedia Layer (SDL) [105]. Existeixen altres alternatives com Allegro [7], però no estan presents a tantes plataformes com SDL. En aquesta secció farem un breu estudi de SDL per conèixer una mica més la llibreria amb la que treballarem.

El nostre joc també usa una altra llibreria multi-plataforma, la libxml2, que s'explica després de SDL, a la secció 3.4.2.

3.4.1 Simple DirectMedia Layer

SDL és una llibreria multimèdia, multi-plataforma i lliure, sota llicència LGPL [46]. Presenta una interfície senzilla per accedir a gràfics, só i dispositius d'entrada de vàries plataformes. S'usa majoritàriament per escriure videojocs o altres aplicacions multimèdia que puguin ser executats en moltes plataformes diferents. El seu nom fa referència a aquesta capa d'abstracció multimèdia simple. Es va començar a desenvolupar l'any 1998 per Sam Lantinga [106].

Per exemple, utilitzant SDL l'àudio es programarà igual per a un PC amb GNU/Linux que per a una Nintendo DS o un iPhone. Això és perfecte per al nostre projecte, perquè haurem de fer un mínim de modificacions al codi del videojoc a l'hora de portar-lo.

La llibreria té per propòsit proporcionar accés transparent a baix nivell de diversos dispositius hardware, com per exemple teclat, ratolí, targeta de só, targeta de vídeo (framebuffer 2D nativament, 3D amb extensions d'OpenGL).

SDL està disponible de manera oficial per GNU/Linux, Windows, Windows CE, BeOS, Mac OS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX. El codi també conté suport per AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS, and OS/2, encara que aquests no són suportats oficialment. Per últim, existeixen ports de tercers per a plataformes com Nintendo DS, iPhone, PlayStation 2, PlayStation Portable, GP32, GP2X, entre d'altres.

SDL està escrita en C, però funciona nativament amb C++, i té interfícies a diversos llenguatges, incloent Ada, C#, D, Eiffel, Erlang, Euphoria, Guile, Haskell, Java, Lisp, Lua, ML, Objective C, Pascal, Perl, PHP, Pike, Pliant, Python, Ruby, Smalltalk, i Tcl.

La llibreria principal es divideix en diversos subsistemes, concretament els subsistemes

de vídeo (tant funcions bàsiques com les de OpenGL), àudio, CD-ROM, joystick, events, threads i rellotge. A més, també existeixen altres mòduls que ofereixen funcionalitats addicionals. Aquests es proporcionen a la web oficial i s'inclouen en la documentació oficial:

SDL_image estén les capacitats per treballar amb diferents formats d'imatge (BMP, JPEG, PNG...)

SDL_mixer estén les capacitats per la gestió i ús de só i música, com per exemple barreja de só. Suporta formats com Wave, MP3, MOD, S3M...

SDL_net proporciona funcions i tipus de dades multi-plataforma per aplicacions que treballin amb la xarxa.

SDL_ttf suport per renderitzat de fonts TrueType

SDL_rtf suport per renderitzar arxius de text amb format Rich Text Format (RTF)

En aquest projecte usarem SDL per fer dibuix 2D, carregar imatges, gestionar events, reproduir àudio i per funcions de xarxa. S'ha treballat amb els mòduls SDL_mixer i SDL_net, a més de la llibreria bàsica.

Actualment existeixen dues versions de SDL, la 1.2 que és la estable, i la 1.3 que encara està en desenvolupament. SDL 1.3 suposa una actualització important del codi base de la SDL 1.2. Substitueix diverses parts de l'API de la versió 1.2 oferint un suport més general per a l'entrada múltiple i opcions de sortida. Algunes novetats a destacar són el suport per a múltiples finestres, suport de múltiples dispositius d'entrada (varis ratolins, teclats, etc), acceleració hardware 3D o suport per a OpenGL 3.0+. També introdueix una nova opció de llicència, a part de la LGPL ofereix una llicència comercial per permetre que codis tancats l'enllacin estàticament. Això obre possibilitats de distribució de la llibreria a plataformes que no permeten l'enllaçat dinàmic. Galaxy Gameworks s'encarrega de la gestió d'aquesta nova llicència [37].

3.4.2 libxml2

La llibreria libxml2 [64] serveix per analitzar documents XML [118]. Està escrita en llenguatge de programació C i és altament portable ja que només depèn de la llibreria estàndard de C. El número 2 del nom fa referència a que va existir una versió prèvia, actualment en desús, que tenia una interfície de programació diferent.

El joc que hem triat l'usa per carregar un fitxer XML que conté el menú d'opcions.

3.5 Eines de compilació

Els entorns de compilació creuada que estem utilitzant es basen en les eines que es descriuen a continuació.

Coneixent aquestes eines podem preparar un entorn de compilació creuada nosaltres mateixos en comptes d'usar un toolchain ja prefabricat. Però cal dir que aquest procediment no és trivial i ens caldrien moltes proves per donar amb un entorn funcional.

Per crear un entorn de compilació creuada necessitarem les eines Binutils [43], GCC i una llibreria de C.

Binutils És una col·lecció d'eines de programació per a la manipulació d'objectes. Les principals eines són l'enllaçador (`ld`) i l'assemblador (`as`).

GCC El compilador de GNU, del que hem parlat a la introducció d'aquest capítol 3 suporta vàries plataformes i llenguatges de programació, i pot ser configurat com a un compilador creuat. En el moment d'escriure aquestes línies l'última versió estable de GCC és la 4.5.2.

Llibreria de C Necessitarem una implementació de les crides a sistema i funcions que permetin reservar i alliberar memòria o gestionar l'entrada i sortida del programari, com per exemple: open, malloc, free, printf o exit. Existeixen vàries implementacions d'aquesta llibreria. La estàndard és la glibc [44], encara que hi ha versions més reduïdes com la newlib [71] pensades per sistemes encastats.

Unes altres eines a destacar són les autotools (GNU build system), que serveixen per automatitzar la creació dels scripts de configuració de paquets i la compilació. Les eines que conté aquest paquet són autoconf, automake i libtool.

Autotools usen les següents notacions:

build platform Per referir-se a on es compila el codi.

host platform Per referir-se a on s'executa el codi.

target platform Per referir-se als compiladors, es refereix al tipus d'objecte que es produirà, com per exemple, compilar un compilador creuat.

De totes formes, aquestes eines s'han d'usar amb cautela per fer compilació creuada, ja que no són gaire fiables per obtenir les dades d'un entorn diferent al natiu del sistema, i les hem de proporcionar nosaltres.

Generalment el codi font que es distribueix inclou un fitxer de configuració anomenat `configure`, algunes vegades, a través d'aquest fitxer (mitjançant paràmetres) es poden generar `Makefile` per a altres plataformes, però no sempre és així.

3.6 Llibreries per a Nintendo DS

A part de l'entorn bàsic de programació, hi ha una gran varietat de llibreries disponibles per diferents propòsits que serveixen d'una o altra manera a la programació per Nintendo DS. A continuació es descriuen les principals llibreries amb l'objectiu d'estudiar quines poden ser útils pel nostre projecte.

SDL ja ha estat presentada a la secció 3.4.1. És una de les llibreries més utilitzades a l'hora de fer videojocs per a PC i ha estat portada a la Nintendo DS. Concretament existeixen dos ports, tots dos no oficials, fets de manera personal. El primer és el de Troy Davis, també anomenat GPF, que és un port de la versió 1.2.9 de SDL [49]; i l'altre fet per Darren Alton pel Google Summer of Code del 2008, que porta la versió 1.3 de SDL [47]. Tots dos treballen sobre devkitARM i libnds.

Aquesta capa d'abstracció és molt útil per portar aplicacions a plataformes menys conegudes com per exemple la Nintendo DS, però en el nostre cas hi ha algun problema: la implementació de SDL per Nintendo DS no està completa, usa una gran quantitat de memòria en relació amb la que hi ha disponible i una gran quantitat de l'acceleració del hardware es veu anul·lada a causa de la capa d'abstracció. Per tant, és una llibreria útil quant a facilitat per a portar aplicacions directament, però s'ha d'anar amb compte per temes de rendiment. En aquest projecte la usarem, tot i aquestes limitacions exposades i conegudes.

dswifi [31] és una llibreria que ofereix una interfície de baix nivell per a la connectivitat DS Wifi. Gràcies a aquesta llibreria podem controlar fàcilment punts d'accés, connexions, adreces IP i paquets. Permet usar connectivitat WiFi estàndard (IEEE 802.11). La llibreria ofereix funcions que permeten que ens connectem a un punt d'accés (que només pot tenir encriptació WEP) i un cop connectats podem utilitzar funcions estàndard de comunicació. Actualment aquesta llibreria també pertany al paquet devkitPro, garantia de que és àmpliament utilitzada, actualitzada i compatible amb libnds. Va ser desenvolupada per Stephen Stair, també conegut com akkit.

Els models Nintendo DSi i Nintendo DSi XL ja permeten encriptació WPA/WPA2, però encara no s'ha implementat aquesta funcionalitat en la llibreria dswifi [30].

Per una altra banda, Nintendo ofereix un altre tipus de comunicació sense fil que permet comunicar dues o més consoles Nintendo DS sense necessitat de passar per un punt d'accés (mode *ad hoc*). Aquest protocol propietari de Nintendo es coneix com Nifi. Actualment cap llibreria homebrew permet comunicar-se usant un protocol similar a aquest, o sense haver de passar per un punt d'accés.

libfat [3] és una llibrería que ens permet accedir al sistema de fitxers. Això és útil sobretot per llegir fitxers de recursos (imatges, sons, etc.) i per escriure configuracions, partides desades, etc. Com hem vist a l'apartat 2.1.3, els flashcart ofereixen un sistema de fitxers subjacent, essent el més comú el FAT16. Per manegar aquest sistema de

fitxers es pot usar libfat. Aquesta depèn de libnds, i també forma part del paquet devkitPro. Amb libfat els fitxers poden ser gestionats de manera quasi exacta a com es fa amb els sistemes POSIX. Això la fa una llibreria molt còmoda d'utilitzar, donat que no fa falta quasi aprenentatge. Tanmateix els directoris segueixen convencions diferents. Va ser creada per Michael "Chishm" Chisholm i és successora de la llibreria GBA NDS FAT.

Cada model de flashcart utilitza el seu propi driver per accedir al sistema de fitxers, i per tant unes comandes de lectura i escriptura diferents; però aquestes comandes no són les mateixes que les específiques de libfat. Per solucionar això existeix una tecnologia anomenada *DLDI* (Dynamically-Linked Device Interface) [26], que consisteix en adjuntar una interfície al programa homebrew de manera que gràcies a aquesta interfície pugui existir comunicació entre les comandes de lectura i escriptura del flashcart i les de la libfat. Aquesta tecnologia també ha estat desenvolupada per Michael "Chishm" Chisholm. A les últimes versions de flashcarts de Slot 1, com per exemple R4 DS o M3DS Real, aproximadament a partir de 2007, ja no és necessari utilitzar la interfície DLDI, les targetes automàticament apliquen la interfície DLDI corresponent a l'inici per eliminar la necesitat de fer-ho manualment.

EFS Lib [33] (Embedded File System Library), és una llibreria que permet accedir a sistemes de fitxers encastats (embedded). Existeix un tipus de sistema de fitxers anomenat NitroFS, propietat de Nintendo, que és com es guarden les ROMs [1] comercials, i que implica que totes les dades estiguin guardades en un únic fitxer `.nds`. Això pot ser útil per a que l'aplicació sigui autònoma, no necessiti fitxers externs, en contraposició a la libfat. Un altre avantatge és que permet carregar més de 4Mb de dades, ja que les dades guardades al sistema de fitxers encastat no s'han de carregar en memòria RAM. La llibreria EFS Lib és una de les opcions que existeixen per manegar sistemes de fitxers NitroFS.

Es necessitarà la llibreria libfat per poder localitzar i accedir a la ROM. La creació i l'annexat del sistema de fitxers es fa amb l'eina `ndstool`, inclosa al devkitARM. L'arxiu `.nds` resultant ha de ser passat per un programa `efs.exe`, per tal de reservar una mica d'espai a la capçalera de la ROM per ubicar una memòria cau, per buscar després al sistema de fitxers adjuntat.

Nitrofs [82] és una altra llibreria que permet accedir al sistema de fitxers encastat (embedded), igual que EFS Lib. Actualment forma part del paquet devkitPro (amb el nom de libfilesystem) la qual cosa és un gran avantatge, ja que és mantinguda i millorada constantment. A més, Nitrofs no requereix un programa extern (efs.exe), i té més compatibilitat amb emuladors. Per últim, inclou "suport argv", que permet passar (a les flashcarts que ho implementin [53]) la localització de la ROM que és executada, de manera que no s'ha de llegir tota la targeta buscant-la, com passa amb

[1] Una imatge ROM, o simplement ROM, és un arxiu d'ordinador que conté una còpia de les dades d'un xip de memòria ROM (en anglès Read-Only Memory), sovint de cartutxos de videojocs o del tauler de comandament d'una màquina recreativa.

EFS Lib. Per tots aquests motius, les principals llibreries d'alt nivell, com PAlib o NightFox's Lib, han anat passant el seu suport d'EFS Lib a Nitrofs [72].

PAlib [88] (amb significat Programmer's Arsenal) és una llibreria que ofereix funcionalitats d'alt nivell, abstraient al programador del hardware. Està construïda sobre libnds, i el que fa és encapsular operacions relacionades amb el hardware en funcions i estructures del llenguatge d'alt nivell C. Els seus mètodes estalvien temps al programador i faciliten la programació (sobretot als principiants), però alguns puristes del hardware consideren que és restrictiva i que es perd rendiment (en comparació amb libnds). Com s'anuncia a la seva pròpia pàgina oficial, des del 7 de juliol de 2010 està abandonada i es busca alguna persona que la continuï mantenint. Tot i així, és una llibreria molt estesa i utilitzada, com demostra la gran quantitat de projectes que hi ha per la xarxa creats amb aquesta llibreria [90].

NightFox's Lib [73]. Existeixen més llibreries d'alt nivell, encara que PAlib és la més usada i coneguda. Per exemple estan les NightFox's Lib, que també treballen sobre devkitPro, i ofereixen funcions WiFi, dibuix de fons, de textos, càrrega de sprites, suport a sistemes de fitxers (FAT y NitroFS), sistema de col·lisions, etc. Disposa d'actualitzacions constants i d'un bon suport per part de l'autor, que a més és de parla hispana. És una bona alternativa que és bo conèixer. El projecte està sota llicència Creative Commons [16], pel que s'ha de tenir cura, ja que a l'hora de publicar haurem d'indicar la seva autoria i no podem treure benefici material del seu ús.

3.7 Emuladors de Nintendo DS

Quan s'està fent compilació creuada, el resultat es pot provar en la màquina destí o directament en la màquina en què es programa mitjançant un emulador. Això estalvia temps en algunes proves. Ens interessa estudiar els emuladors que hi ha disponibles per conèixer les seves característiques i poder escollir el que millor s'adapta a les nostres condicions i necessitats. Podem trobar part d'aquesta informació a [77, 78].

DeSmuME [22] (també conegut com DS YopYop) és un emulador lliure per a la Nintendo DS creat per YopYop156. Originalment estava escrit per a Windows i en francès. Al voltant de 2006, el projecte es va convertir en programari lliure (llicència GPL) i van sortir també les versions per a GNU/Linux i Mac OS, i en altres idiomes. Les versions posteriors contenen múltiples interfícies d'usuari i són multi-plataforma, arribant a existir una versió fins i tot per a la consola PlayStation Portable (la qual no te pantalla tàctil). YopYop va aturar el desenvolupament de DeSmuME a causa de les lleis relatives a l'emulació en França; no obstant això, altres programadors van seguir el seu desenvolupament.

El nom és un joc de paraules de les paraules DS, emu de emulador, i ME de l'ús

popular de la paraula ME entre els desenvolupadors de homebrew (PassMe, WifiMe, FlashMe).

Totes les característiques 2D conegudes estan suportades. Les característiques 3D són emulades excepte antialiasing, i algunes vegades falla el renderitzador d'OpenGL. Les característiques de só són emulades excepte el micròfon, que només està emulat en Windows. La WiFi no està suportada. Quant a accés a sistema de fitxers, té suport per libfat (hem de preparar la ROM amb el driver DLDI del flashcart "GBA Movie Player versió Compact Flash" sinó no funcionarà), però no té suport per sistemes de fitxers embedits (EFS o NitroFS). Segons algunes fonts com [77], referent a emulació de ROMs comercials, DeSmuME té més compatibilitat que els altres emuladors.

No permet depurar codi directament, però es pot integrar amb el depurador GDB (GNU Debugger) durant un programa en execució, establir punts d'interrupció, inspeccionar el contingut de la memòria i tot el que fa un depurador.

L'última versió és de maig de 2010, pel que sembla que no està molt abandonat.

iDeaS [56] és un emulador gratuït, però no lliure (freeware) que té versions per Windows i GNU/Linux. A Linux necessita per funcionar: GTK+ (conjunt de llibreries multi-plataforma per desenvolupar interfícies gràfiques d'usuari) i dues llibreries que pertanyen a aquest entorn, Pango (per dibuix de textos) i Cairo (per gràfics vectorials).

Emula el processador ARM7 al 100% i el processador ARM9 al 99%, permetent executar moltes ROMs comercials. La pantalla tàctil està totalment suportada, a traves de ratolí. Micròfon parcialment emulat. WiFi funciona per algunes aplicacions homebrew. Existeix una versió de l'emulador que inclou debugger per programadors.

Tot i que el codi no és obert, el fòrum d'iDeaS té una secció d'informe d'errors de manera que els usuaris poden enviar problemes que hagin trobat en jocs. A més, com que l'emulador està basat en complements (plugin), els usuaris poden programar els seus propis complements per solucionar aquests problemes.

L'última versió és d'octubre del 2010, pel que està bastant actualitzat.

No$GBA [83] (llegit no-cash-gba) és un altre emulador freeware, però només per Microsoft Windows i DOS. Es pot utilitzar en GNU/Linux mitjançant Wine, una aplicació que permet executar aplicacions de Windows en GNU/Linux i altres entorns [116]. És desenvolupat per Martin Korth. És capaç d'executar jocs comercials i homebrew de Nintendo DS i de GameBoy Advance. És considerat un dels millors emuladors de Nintendo DS perquè és capaç d'executar molts jocs comercials a velocitat màxima i sense errors de vídeo o àudio. És considerat per molts el primer emulador en executar ROMs comercials (sobre l'agost de 2006) [84].

Martin Korth no ha publicat una nova versió des de gener de 2008, i no s'ha sabut res més d'ell des de llavors, pel que es pressuposa que el projecte està aturat, encara

que el suport dels fans continua. Té suport per gràfics 2D, 3D, só, micròfon i WiFi (parcialment).

L'emulador es manté freeware fins a la versió 2.6, per obtenir la versió 2.6a (l'última) s'ha de fer una "donació" de 2.5 dòlars. També existeix una versió de l'emulador orientada per a desenvolupadors, amb la que es poden fer proves i depuració, però no és freeware, s'ha de pagar d'acord a la versió del depurador que es vol (des dels 15 dòlars fins als 5000 dòlars).

Existeixen altres opcions, però que no són tan interessants per al nostre projecte, pel que no farem un estudi tan intensiu. Els motius són que no tenen tan avançada la emulació de la consola, que fa temps que no estan en desenvolupament, que no tenen llicències lliures, i que només funcionen sota Windows. Alguns exemples d'altres emuladors, per si el lector està interessat, són Dualis [32], Ensata [34], i els menys desenvolupats DSEmu [29] i NeonDS [70].

Després d'aquest estudi, decidim utilitzar DeSmuMe, per ser programari lliure, per tenir versió per a GNU/Linux, per la seva gran capacitat d'emulació de les capacitats de la consola, i per la possibilitat de depurar el codi.

Capítol 4

Anàlisi del joc

Per acabar amb la fase d'anàlisi, abans de començar amb la implementació del projecte, necessitàvem un joc que portar a les respectives plataformes. En un primer moment vam pensar en desenvolupar un videojoc nosaltres mateixos, però després vam decidir que aquest no era l'objectiu principal del projecte i vam decidir buscar un joc lliure que s'adaptés a les nostres necessitats, és a dir, que complís les següents condicions:

- Fàcilment portable (preferiblement programat en C/C++ utilitzant les llibreries SDL)
- Lliure
- Opció de joc multi-jugador en xarxa

Després de molt buscar, vam trobar diferents opcions que s'adaptaven a les nostres necessitats, però finalment ens vam decidir per un joc anomenat 2Pong. El vam escollir perquè complia totes les característiques que demanàvem, i per la seva simplicitat.

El 2Pong [1] es tracta d'un clon del mític Pong [94], considerat per molts com el primer videojoc de la història. Pel qui no el conegui, és un joc en dues dimensions que simula un tennis de taula. El jugador controla una paleta que es mou de dalt a baix, i competeix contra una altra paleta, que pot estar controlada per la computadora o per un altre jugador. Els jugadors usen aquestes paletes per fer rebotar una bola d'un costat a l'altre. Cada cop que un jugador deixa passar la bola, fa guanyar un punt a l'adversari.

La peculiaritat del 2Pong és que es juga amb dues boles en comptes d'una sola, d'aquí el seu nom. És un joc molt senzill. Consta d'un menú, en el que es pot escollir jugar un sol jugador contra la intel·ligència artificial (amb tres nivells de dificultat diferents), i el mode multi-jugador en xarxa, en el que el jugador pot escollir entre fer de servidor o de client, i que es comunica via una IP i un port. Pel que fa a l'àudio, no té música d'ambient, només te efectes de só quan rebota la bola. Per últim, remarcar que té uns "powerups", que són

uns objectes modificadors que apareixen a la pantalla i que tenen diversos efectes sobre la bola: augmentar la seva velocitat, disminuir-la, dividir la bola en dues, etc.

El 2Pong està escrit en C++ i utilitza les següents llibreries: SDL, SDL_net, SDL_mixer, iconv, zlib i libxml2. Per tant, si volem compilar el codi font, haurem de preparar el nostre sistema perquè suporti totes aquestes llibreries.

L'última versió és la 1.0.1a i data del maig de 2005, pel que fa bastant temps que el projecte no està en desenvolupament, encara que s'anunciaven millores per a la següent versió. A la web oficial del projecte estan disponibles els binaris de la darrera versió per a GNU/Linux i Windows.

El projecte es distribueix sota llicència GPL [45], que permet la còpia, distribució (comercial o no) i modificació del codi, sempre que qualsevol modificació es continuï distribuint amb la mateixa llicència GPL. La llicència GPL no permet la distribució de programes executables sense el codi font corresponent o la indicació de com obtenir-lo gratuïtament. És important conèixer la llicència donat que nosaltres haurem de fer modificacions al codi, i si les volem publicar ho haurem de fer de la manera que ens marca la llicència.

El seu creador és Itay Kirshenbaum i hem contactat amb ell per tal d'informar-lo dels ports realitzats i de la nostra intenció de publicar el codi. No ens ha posat cap problema i ha mostrat interès pel projecte.

4.1 Algorisme

A continuació es resumeixen les fases per les que passa el joc.

1. Inicialització de SDL
2. Creació de la finestra SDL
3. Càrrega dels sprites
4. Creació del menú XML
   (a) Si es tria **regular** (mode humà vs CPU), elecció de la dificultat
   (b) Si es tria **network** (mode humà vs humà), elecció de client/server
      i. Especificar IP i port a connectar
5. Inici del joc (bucle principal)
   (a) Captura i tractament d'events de SDL
   (b) Actualització de l'estat del joc
   (c) Actualització dades xarxa (en cas d'haver triat mode de xarxa)

   (d) Pintat de l'escena

6. Sortida del joc

A la figura 4.1 es mostra l'esquema del menú del 2Pong. S'indiquen les opcions que cal triar per a efectuar les transicions entre pantalles.

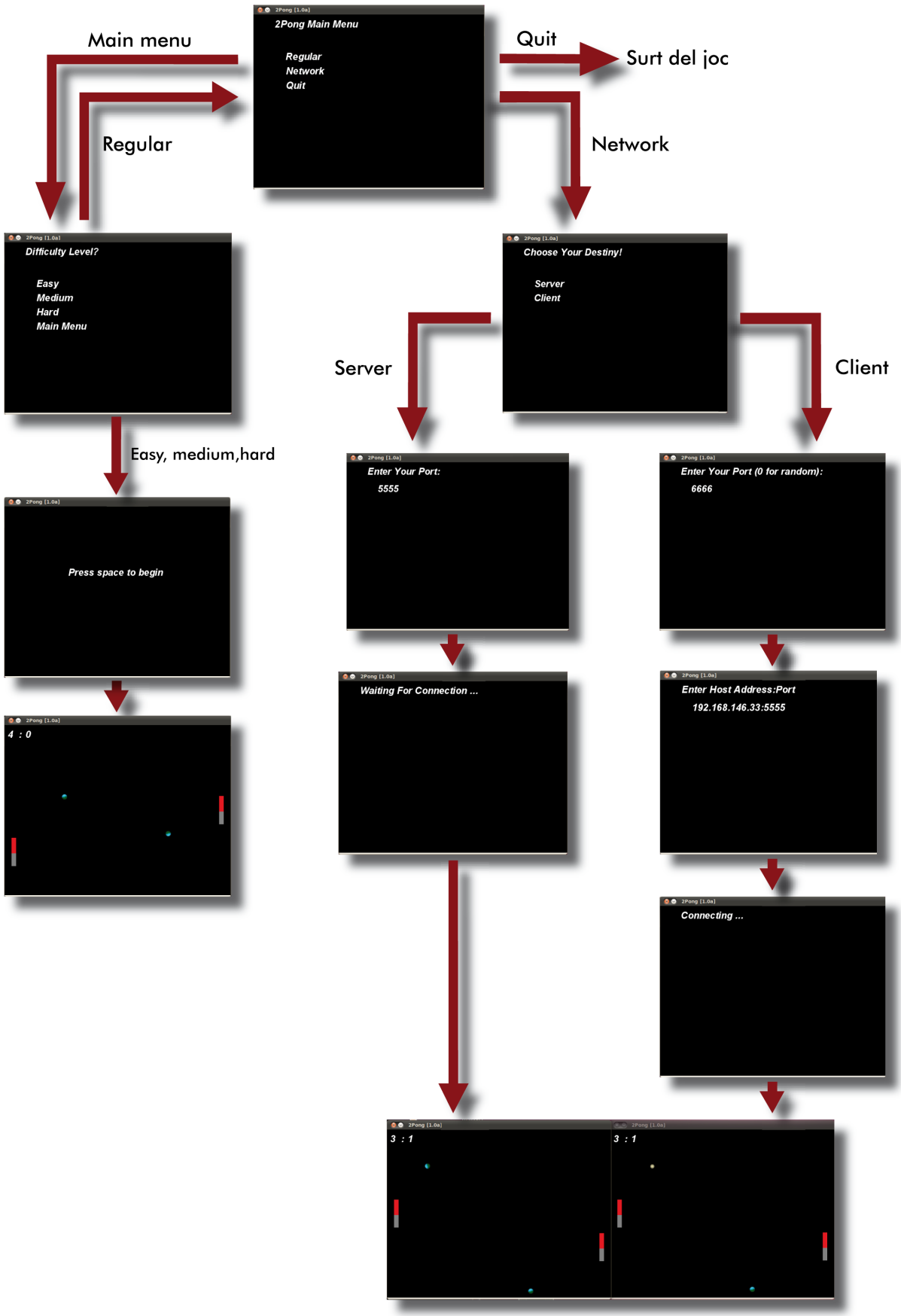

Figura 4.1: Mapa dels menús del 2Pong.

Capítol 5

Implementació

Després de tota la fase d'estudi previ, ja estem preparats per realitzar la implementació del projecte.

Aquest capítol reflecteix tot el treball fet per nosaltres i en ell es detalla tota la programació i els experiments realitzats, de manera que aquests procediments puguin ser reproduïts pel lector.

Hem dividit el procés en diverses fases. Primer explicarem com instal·lar un entorn de desenvolupament com l'explicat a la fase d'anàlisi per a Nintendo DS. A continuació ensenyarem com compilar algunes llibreries que es necessiten per al 2Pong i que no existeixen per a la Nintendo DS. Després, explicarem la instal·lació de l'entorn de desenvolupament pel dispositiu PSP. Tot seguit comentarem els canvis que s'han fet al codi del joc per arreglar-lo, adaptar-lo a les diferents plataformes i compilar-lo. Per últim explicarem com hem contribuït amb tot el codi generat amb la comunitat del programari lliure.

5.1 Instal·lació de l'entorn de desenvolupament de Nintendo DS

En aquest apartat explicarem com instal·lar l'entorn de desenvolupament que s'ha utilitzat en aquest projecte. Començarem per la instal·lació del devkitPro i altres llibreries per Nintendo DS i després explicarem com instal·lar el port de SDL per Nintendo DS. Les altres llibreries les hem deixat per una altra secció, ja que precisen més modificacions en el codi.

5.1.1 Instal·lació del toolchain devkitPro

Com hem vist en el capítol anterior, el toolchain habitual per programar homebrew per Nintendo DS és el devkitPro, amb totes les eines que aquest proporciona: devkitARM, libnds, etc. En aquesta secció veurem com instal·lar un entorn d'aquest estil per poder programar per aquesta plataforma.

La instal·lació per a Windows és trivial, existeix un executable que s'encarrega automàticament de tot el procés d'instal·lació. La instal·lació per a GNU/Linux no és molt més complicada, però requereix alguns coneixements del sistema operatiu, com per exemple modificar variables d'entorn.

Pas 1: Descarregar els arxius necessaris per a la instal·lació. Es poden descarregar tots de [24]. En concret necessitarem l'última versió de devkitARM, libnds, i l'arxiu "default arm7". Opcionalment podem descarregar els exemples, per provar que hem instal·lat bé l'entorn.

Quan descarreguem els binaris (el devkitARM), s'ha de prestar atenció a descarregar els corresponents a l'arquitectura en la que compilarem (arquitectura host), ja que existeixen dues versions (i686 i x86_64) i si intentem compilar amb la versió equivocada ens trobarem amb errors de compilació.

Les versions que vaig instal·lar i amb les que he treballat en aquest projecte són:

- devkitARM_r26-i686-linux.tar.bz2
- libnds-src-1.3.7.tar.bz2
- default_arm7-0.5.6.tar.bz2
- nds-examples-20091201.tar.bz2

En el moment d'escriure aquesta memòria l'última versió de devkitARM és la r32, i ni tan sols es pot descarregar de manera oficial la versió r26, però aquest fet no canvia el procés d'instal·lació explicat en aquesta secció. He seguit aquests mateixos passos amb les últimes versions i no hi ha cap problema. Les últimes versions en el moment d'escriure aquestes línies són:

- devkitARM_r32-i686-linux.tar.bz2
- libnds-src-1.4.8.tar.bz2
- default_arm7-0.5.17.tar.bz2
- nds-examples-20101020.tar.bz2

Pas 2: Crear el directori de treball. Es necessita un directori on instal·lar els binaris i llibreries per tenir-hi accés més endavant. Aquest directori pot ser qualsevol, però en el meu cas concret el vaig instal·lar a:

`/home/usuari/devkitPro`

I en aquest tutorial d'instal·lació utilitzarem aquest directori en tots els exemples. Una altra possibilitat bastant habitual seria crear el directori a:

`/usr/local/devkitPRO`

Encara que amb aquest directori haurem d'anar més amb compte amb els permisos del sistema. De tota manera, un cop finalitzada la instal·lació, després de seguir tots els passos, ens ha de quedar una estructura com la següent:

```
.../devkitPro
.../devkitPro/devkitARM/
.../devkitPro/libnds/
```

Per tant, per crear el directori fem:

```
mkdir /home/usuari/devkitPro
```

Evidentment, canviant `usuari` pel nostre nom d'usuari.

Pas 3: Instal·lar els binaris de devkitPro. Només cal descomprimir dins el directori `devkitPro` l'arxiu que havíem descarregat.

```
tar -xjvf devkitARM_r26-i686-linux.tar.bz2
```

Això crearà un directori devkitARM amb el toolchain, que entre d'altres coses porta els binaris necessaris per compilar per Nintendo DS (arm-eabi-gcc, ndstool, etc.)

Pas 4: Configurar l'entorn. Abans de compilar la llibreria libnds necessitem crear algunes variables d'entorn, de manera que els makefiles sàpiguen on estan instal·lats els binaris. Això ho farem amb les següents comandes:

```
export DEVKITPRO="/home/usuari/devkitPro"
export DEVKITARM="$DEVKITPRO/devkitARM"
export PATH="$DEVKITARM/bin:$PATH"
```

Si volem que aquestes variables d'entorn siguin accessibles per sempre, a més haurem de copiar aquestes dues mateixes línies al final del fitxer "`~/.bashrc`". Per fer que els canvis

del fitxer tinguin efecte immediat en el sistema executem "`source ~/.bashrc`". Podem comprovar que s'han guardat correctament executant:

```
env | grep DEVKITARM
```

O bé

```
echo $DEVKITARM
```

Pas 5: Instal·lar la llibreria libnds. Primer necessitem descomprimir l'arxiu de la llibreria libnds que hem descarregat. Però a diferència de l'arxiu devkitARM, aquest no conté un directori base, l'haurem de crear manualment.

```
mkdir $DEVKITPRO/libnds
mv libnds-src-1.3.7.tar.bz2 $DEVKITPRO/libnds
cd $DEVKITPRO/libnds
tar -xjvf libnds-src-1.3.7.tar.bz2
```

Ara haurem de compilar la llibreria amb la comanda `make` i si tot ha anat bé, s'instal·larà amb èxit.

Per últim, haurem de comprovar si al directori "devkitPro/libnds" existeix un arxiu "default.arm7". Aquest arxiu és necessari per compilar per Nintendo DS. Com ja hem vist anteriorment, els programes de Nintendo DS consten de dos executables: un pel processador ARM9 (el principal) i un altre per l'ARM7 (normalment un programa secundari que s'encarrega de tasques com el só, la pantalla tàctil o la Wi-Fi). Si no volem fer nosaltres un programa per aquest processador, s'utilitzarà aquest executable per omissió, el `default.arm7`. En versions més antigues de libnds, aquest venia inclòs en la llibreria libnds, però ara l'hem de copiar manualment. Per fer-ho només s'ha de descomprimir l'arxiu `default_arm7-0.5.6.tar.bz2` i copiar el contingut a `devkitPro/libnds`. També existeix una versió sense compilar d'aquest programa, per si volem veure o modificar el codi, o compilar-lo nosaltres mateixos; però abans de poder compilar-lo haurem d'instal·lar les llibreries dswifi i maxmod, perquè aquest les utilitza.

Pas 6: Provar la instal·lació. Amb això ja tenim un entorn de desenvolupament bàsic disponible. Ja es poden compilar els exemples oficials de devkitPro per provar que l'entorn s'ha instal·lat satisfactòriament. Aquests porten un makefile que s'encarrega de compilar tots els exemples disponibles, però hem de tenir en compte que ara mateix no compilaran tots, degut a que alguns utilitzen funcionalitats que encara no hem instal·lat. Per exemple els tests que hi ha al directori dswifi no compilaran perquè no hem instal·lat la llibreria dswifi. Podem provar per exemple a compilar el test hello_world, que és el més bàsic, i amb això ja sabrem si hem instal·lat l'entorn correctament.

Per acabar, ja podem eliminar tots els arxius comprimits `.tar.bz2` que havíem descarregat i que ja estan instal·lats.

Pas 7: Instal·lar llibreries addicionals. Com ja hem dit, amb aquesta instal·lació tenim un entorn de desenvolupament bàsic de Nintendo DS. Amb això ja tenim la possibilitat de compilar alguns exemples, i també la llibreria SDL. Tot i això és interessant instal·lar altres complements, com per exemple la llibreria libfat, que serveix per utilitzar el sistema de fitxers, o la llibreria dswifi, que ens permet accedir a les funcions de xarxa. Aquest projecte en concret utilitza les dues ja que necessitarem tant accés al sistema de fitxers com connexió a la xarxa per al nostre joc. Com hem estudiat anteriorment, aquestes dues llibreries pertanyen també al paquet devkitPro.

Per instal·lar aquestes llibreries haurem de seguir un procés molt semblant al seguit fins ara. Donat que pertanyen al paquet devkitPro, també es poden descarregar de citedevkitpro:sourceforge. Les versions que jo vaig instal·lar són:

- libfat-src-1.0.6.tar.bz2
- dswifi-src-0.3.11.tar.bz2

I les versions disponibles en el moment d'escriure la memòria són:

- libfat-src-1.0.7.tar.bz2
- dswifi-src-0.3.13.tar.bz2

Un cop descarregades, fem `make nds-install` per la libfat, i `make install` per la dswifi. Això ja ens ho compilarà i copiarà tots els arxius necessaris als directoris corresponents de libnds.

I amb això ja tenim un entorn més complet amb el que podrem compilar la majoria dels exemples oficials de devkitPro, la llibreria PAlib, la llibreria SDL, etc.

5.1.2 Instal·lació de la llibreria d'alt nivell PAlib

En un principi ens va semblar interessant instal·lar la llibreria PAlib (introduïda a 3.6) ja que facilitaria la programació per Nintendo DS que s'hagués de fer. El problema és que faria el projecte més depenent de la plataforma, perquè obligaria a instal·lar llibreries addicionals al toolchain, i per això finalment no s'ha utilitzat.

El problema és que, encara que no ho sembli, instal·lar PAlib no és trivial donat que funciona sobre libnds però són dos projectes totalment independents i a vegades surten incompatibilitats entre les dues segons les versions que s'utilitzin.

No s'explicarà tot el procés d'instal·lació de PAlib, que es pot consultar a la seva wiki [89], només es vol comentar una combinació de versions que es va aconseguir fer funcionar, per facilitar la feina al lector en cas que la vulgui instal·lar. Si volem trobar una combinació de versions que funcioni ens convé rebaixar la versió de PAlib i no la de libnds, ja que aquesta és més important perquè és la llibreria de sistema.

No obstant, la única solució no és trobar una combinació de versions que funcioni, evidentment també podem actualitzar el codi de PAlib per a que sigui compatible amb la última versió de libnds, però és més costós.

En el moment en que jo vaig instal·lar, la versió oficial de PAlib del moment no era compatible amb les últimes versions de devkitPro. Per això alguns usuaris van preparar una versió compatible i la van distribuir per separat. Aquesta versió s'anomena 090621 i es pot obtenir de [87]. Jo puc assegurar que aquesta versió és compatible amb devkitARM r26, libnds 1.3.7, dswifi 0.3.10 i libfat-nds 1.0.6.

5.1.3 Instal·lació del port de SDL

Vam introduir les llibreries SDL a l'apartat 3.4.1 i vam explicar que hi havia dos ports realitzats per a Nintendo DS a l'apartat 3.6.

Després de provar els dos ports amb alguns exemples senzills de SDL, ens vam decidir per utilitzar el de Troy Davis, ja que és el que donava més bons resultats. En tots dos hi havien molts exemples que no compilaven perquè hi ha moltes funcionalitats que no han estat portades, però amb el de Troy Davis hi van haver més exemples que van compilar.

A continuació es detalla el procés d'instal·lació.

Pas 1: Descarregar codi font. El codi es pot descarregar del citat lloc [49]. L'última versió és del 15 de juny del 2007 (per tant podem afirmar que és un projecte abandonat) i és compatible amb devkitARM r20. Com ja hem dit, l'última versió de devkitARM en el moment d'escriure aquestes línies és la r32, per tant està bastant desfasat. De fet, s'han de fer modificacions al codi font per poder-lo utilitzar amb les últimes versions de devkitARM.

Existeixen dues versions del port, una precompilada i una amb el codi font. Però com que haurem de fer modificacions en el codi per fer-la compatible amb les últimes versions de devkitARM, no podrem utilitzar la versió precompilada. Un cop descarregat el codi, descomprimim el paquet. Necessitem tenir el paquet rar instal·lat. Descomprimim amb la comanda:

```
rar x SDL-1.2.9DS061507src.rar
```

Pas 2: Fer modificacions al codi font. Aquest pas serà diferent segons quina versió de devkitARM estiguem utilitzant. Per exemple, si utilitzem devkitARM r20, no necessitarem fer cap canvi perquè és compatible amb el port de SDL. Però aquesta versió del devkitARM ja no és fàcil d'aconseguir, a banda de que no és aconsellable, donat que està obsoleta.

En canvi, si volem compilar amb un devkitARM r26, com el que vaig instal·lar jo, haurem de fer aquestes petites modificacions:

- Hi ha una funció que va canviar de nom. Haurem d'editar l'arxiu:

  `SDL-1.2.9/src/video/nds/SDL_ndsvideo.c`

  i a la línia 200 modificar la crida a:

  ```
  powerON(POWER_ALL);
  ```

  per:

  ```
  powerOn(POWER_ALL);
  ```

- Hi ha una funció que va desaparèixer i l'haurem de substituir per una altra. Haurem d'editar el mateix arxiu `SDL_ndsvideo.c` d'abans i a la línia 248 canviar la crida a:

  ```
  consoleInitDefault((u16*)SCREEN_BASE_BLOCK_SUB(8),
  (u16*)CHAR_BASE_BLOCK_SUB(0), 16);
  ```

  per:

  ```
  consoleDemoInit();
  ```

Per últim, per fer-ho compatible amb les últimes versions de devkitARM com la r32 amb la que també s'ha provat s'hauran de fer uns últims canvis addicionals. Això és degut a que ha desaparegut de la llibreria libnds una estructura anomenada IPC que s'utilitzava en el port de SDL. No és que aquest codi doni problemes quan compilem la llibreria, ja que aquesta compilarà perfectament; però sí que ens donarà errors de compilació quan utilitzem certes funcionalitats de la llibreria SDL. Per tant, hem d'evitar aquests futurs problemes.

Per solucionar-ho tindríem vàries opcions. La més correcta seria substituir aquesta funcionalitat que ha deixat d'existir per una d'equivalent. Però en aquesta ocasió optarem per la solució més senzilla, que és eliminar aquesta funcionalitat, comentant qualsevol aparició de la desapareguda estructura IPC. Haurem de comentar algunes funcions dels fitxers `src/audio/nds/SDL_audio.c` i `src/audio/nds/audio9.c`, en concret el contingut de les funcions `MixSound()` de `SDL_audio.c`; i `SoundSystemInit()` i `soundstartmixer()` de `sound9.c`. Òbviament, el fet de comentar aquestes funcions farà que perdem funcionalitat, en aquest cas l'àudio de SDL. Però al nostre cas en concret no ens afectarà, perquè el joc utilitza el mòdul SDL_mixer per a la part d'àudio i per tant no utilitza aquestes funcions.

Pas 3: Compilar. I amb aquests canvis, ja està el codi preparat per compilar. Anem al directori `SDL-1.2.9` i escrivim

```
make -f Makefile.ds
```

Faltarà copiar els includes. Per copiar-los fem:

```
mkdir $DEVKITPRO/libnds/include/SDL/
cp include/* $DEVKITPRO/libnds/include/SDL/
```

Podem provar que s'hagi instal·lat bé compilant els exemples que es poden descarregar també a la pàgina de GPF [49]. L'arxiu s'anomena `NDSSDLexamples.rar`, i en descomprimir, dóna un error en extreure un arxiu, pel que és probable que un dels exemples no funcioni, però la resta s'extreuen bé. Tot i això, encara no serem capaços de compilar tots els exemples de SDL, ja que alguns utilitzen altres mòduls que encara no hem instal·lat (`meSDL` utilitza `SDL_image`, `ttfSDL` necessita `SDL_ttf`, etc.). Sí que podem compilar els exemples fireSDL, joytestsdl, rectSDL i sdl_wingman.

Instal·lant SDL_mixer

Per poder portar el 2Pong també necessitarem instal·lar el mòdul SDL_mixer, que també ha estat portat per Troy Davis. És una instal·lació molt senzilla, encara que també haurem de compilar els fonts. Per fer-ho haurem de descarregar el fitxer `SDL_mixer-1.2.6ds.rar` del lloc de GPF [49]. Desempaquetem el fitxer amb la comanda:

```
rar x SDL_mixer-1.2.6ds.rar
```

En aquesta ocasió abans de compilar copiarem els includes, ja que alguns són necessaris per la compilació. Des del directori principal executem:

```
cp source/mikmod/*.h source/SDL_mixer.h $DEVKITPRO/libnds/include/SDL/
```

Després ja estem preparats per compilar amb la comanda `make`. A més de la llibreria SDL_mixer, es compilaran dues llibreries addicionals, anomenades mikmod i timidity. En aquest cas, ja s'instal·len les llibreries resultants en els directoris corresponents, no haurem de copiar els resultats.

5.2 Compilant llibreries per Nintendo DS

El 2Pong necessita altres llibreries per poder funcionar: SDL_net i libxml2. Però aquestes llibreries no ens les ofereixen ja preparades per ser compilades per Nintendo DS. En aquesta

secció explicarem com compilar-les nosaltres mateixos per Nintendo DS, i aprofitarem per experimentar diferents mètodes de compilació.

5.2.1 Portar SDL_net a Nintendo DS

A diferència d'altres mòduls de SDL, SDL_net no havia estat portada abans a Nintendo DS, o com a mínim nosaltres no hem trobat cap port a Internet. Això significa que si el volem utilitzar en Nintendo DS l'haurem de portar nosaltres mateixos, el que significa que haurem d'agafar el codi de SDL_net i compilar-lo nosaltres per la plataforma Nintendo DS. Això és possible gràcies a que les funcions estàndard de xarxa que utilitza la SDL_net, com per exemple els sockets, han estat implementades a la llibreria dswifi.

Els passos a seguir per portar la llibreria a Nintendo DS es descriuen a continuació.

Per començar, descarreguem l'última versió de codi de la llibreria de [104], en el meu cas concret l'arxiu `SDL_net-1.2.7.tar.gz`. El desempaquetem amb la comanda:

```
tar xzvf SDL_net-1.2.7.tar.gz
```

El següent pas serà compilar la llibreria per Nintendo DS. Tenim vàries opcions per fer-ho, per exemple amb scratchbox, autotools o amb un Makefile. En aquesta ocasió descriurem aquesta última opció. Per facilitar la tasca i no haver de crear el Makefile, utilitzarem una plantilla per compilar llibreries que hi ha entre els exemples del devkitPro, que ja vam descarregar en el seu moment.

També haurem de fer unes petites modificacions en el codi abans de compilar la llibreria per a que aquesta funcioni correctament. A continuació es detallen els canvis.

Copiar i modificar el Makefile

Si durant la instalació del toolchain, a l'apartat 5.1.1, vam descarregar els exemples, tindrem la plantilla; sinó haurem de descarregar els exemples de devkitPro. Un cop els tinguem, copiem la plantilla al directori base de la llibreria SDL_net que hem descarregat:

```
cp .../nds-examples-20100313/templates/arm9lib/Makefile .../SDL_net-1.2.7
```

I fem aquestes modificacions:

```
TARGET   := SDL_net  # Nom que volem per la llibreria
BUILD    := build    # Directori on es crearan els fitxers objectes (.o)
SOURCES  := .        # Directoris on es troben els fonts (en aquest cas
   el mateix directori que el Makefile)
DATA     :=          # Directoris on es troben fitxers amb dades (en
   aquest cas cap)
INCLUDES := .        # Directori on es troben headers
CFLAGS   += $(INCLUDE) -DARM9 -I$(DEVKITPRO)/libnds/include/SDL -I$(
   DEVKITPRO)/libnds/include
# Afegim als flags de C els paths perque trobi els headers de les
   llibreries libnds i SDL.
```

Exemple que no volem compilar

Juntament amb la llibreria s'inclou un codi d'exemple per provar el seu funcionament un cop compilada. El problema és que si el deixem al mateix directori, el nostre Makefile l'agafarà com a codi de la llibreria i es compilarà com a part d'aquesta. Per tant, ja que no volem que l'exemple formi part de la llibreria, mourem el seu codi a un altre directori. Podem crear per exemple un directori anomenat `chat`, i haurem de moure-hi els fitxers `chat.cpp`, `chatd.c` i `chat.h`.

Capçalera no trobada

Si intentem compilar, ens trobarem amb un missatge d'error indicant que no troba un dels includes:

```
ERROR: fatal error: arpa/inet.h: No such file or directory
```

Això és perquè en instal·lar la dswifi, aquesta capçalera no s'instal·la. Aquest arxiu de capçalera `arpa/inet.h` [12] defineix algunes operacions per Internet (per exemple per manipulació d'adreces), però realment cap de les seves funcions és utilitzada en el codi de SDL_net, pel que podem comentar aquest include sense cap problema i no tindrem errors. Per tant editem l'arxiu SDLnetsys.h i comentem la línia 56, que és on es troba aquest include.

Estructura modificada

Tindrem un altre error al compilar, que és el següent:

```
error: 'struct hostent' has no member named 'h_addr'
```

```
struct hostent
{
  char *h_name;                  // Official name of host.
  char **h_aliases;              // Alias list.
  int h_addrtype;                // Host address type.
  int h_length;                  // Length of address.
  char **h_addr_list;            // List of addresses from name server.
  #define h_addr h_addr_list[0] // Address, for backward compatibility.
};
```

Figura 5.1: Estructura hostent.

Podem veure com està definida l'estructura hostent (definida a netdb.h) a la figura 5.1. Efectivament, la estructura no té cap atribut anomenat `h_addr`, però té aquesta macro al final, que està definida per temes de compatibilitat. El problema és que al `netdb.h` que instal·la per omissió la dswifi no hi és aquest define. Per tant, o bé modifiquem el `netdb.h` i copiem el define, o bé canviem el codi de la SDL_net. Si optem per aquesta solució, hauríem d'editar `SDLnet.c` i canviar, a la línia 347:

`hp->h>_addr`

Per això altre:

`hp->h_addr_list[0]`

Funció gethostbyaddr

Per últim, ens trobarem amb un avís com el següent:

```
warning: implicit declaration of function 'gethostbyaddr'
```

Que significa que el codi de SDL_net està utilitzant una funció que no està declarada ni implementada enlloc, és a dir, que no està inclosa a la llibreria dswifi.

Per tant, o no utilitzem aquesta funció o l'haurem de portar nosaltres mateixos a Nintendo DS, perquè si ho deixem com està, ara compilarà la llibreria SDL_net, però després tindrem errors de compilació en els programes de Nintendo DS que utilitzin aquesta funció.

Si volem saber com portar la funció, està detallat a la seva pròpia secció 5.2.2, donat la gran extensió d'aquesta secció. Si volem saltar-nos aquest pas, podem comentar la crida a `gethostbyaddr()`. Es troba al fitxer `SDLnet.c`, a la línia 373, dintre la funció `SDLNet_ResolveIP()`. Òbviament aquesta funció deixarà de funcionar.

Compilar i provar

Per últim, compilem des del directori base amb la comanda `make`, i a continuació copiem els resultats als directoris adients:

```
cp SDL_net.h $DEVKITPRO/libnds/include/SDL/
cp lib/libSDL_net.a $DEVKITPRO/libnds/lib/
```

Podem provar que la instal·lació hagi estat correcta compilant l'exemple `chat.cpp` que s'inclou amb el codi de la llibreria o amb qualsevol de les demos que es poden descarregar del seu lloc de documentació [103]. L'exemple més senzill és diu `dnr.c` i també prova la resolució DNS. Però en qualsevol cas haurem de fer modificacions per a que funcioni en Nintendo DS, com per exemple crear-li un Makefile especial, o que no utilitzi els paràmetres per línia de comandes (argc i argv).

5.2.2 Portant una funció estàndard de C

La llibreria SDL_net es basa en funcions estàndard de xarxa per realitzar les seves funcions. A l'hora de compilar SDL_net per Nintendo DS ens adonem que totes les funcions estàndard de C que aquesta utilitza han estat portades exceptuant una: la funció `gethostbyaddr()`, utilitzada per `SDLNet_ResolveIP()`, no està implementada per la Nintendo DS.

La llibreria SDL_net no dona error, ens compilarà només mostrant un avís, però després els fonts de Nintendo DS que utilitzin aquesta funció `SDLNet_ResolveIP()` sí que donaran un error en temps de compilació. Per tant, hem de solucionar el problema d'alguna manera.

La solució més correcta és portar aquesta funcionalitat nosaltres mateixos. Així la llibreria SDL_net que haurem portat estarà completa, i a més, contribuirem amb la comunitat completant encara més la llibreria dswifi, la llibreria de xarxa de Nintendo DS. Per tal d'assolir aquest objectiu començarem per estudiar la funció a portar.

La funció té la següent capçalera:

```
struct hostent *gethostbyaddr(const char *addr, int len, int type);
```

Com hem dit, pertany a la llibreria estàndard de C. Donada una adreça IP, retorna tota la informació sobre els hosts relacionats a aquesta adreça [42]. És a dir, és un sistema de resolució de DNS [101] (fa el que s'anomena resolució inversa). La llibreria de resolució de noms (resolver) consta de dues funcions: `gethostbyname()` resol noms en adreces, i pel contrari `gethostbyaddr()` retorna els noms que corresponen a una adreça. Aquesta funció està declarada a l'arxiu `netdb.h`, i està obsoleta des de la introducció de la funció `getnameinfo()`, per la qual s'ha de substituir el seu ús.

Per portar la funció a Nintendo DS, la primera idea va ser la següent: ja que la funció pertany a l'estàndard de C, es podria recórrer a una implementació de la llibreria C, i

provar d'extreure només aquesta funcionalitat. Vaig provar amb les llibreries Newlib [71] i μClibc [111], perquè són implementacions més portables i lleugeres que es solen utilitzar per plataformes d'aquest tipus. Però va ser impossible extreure la funció `gethostbyaddr()`, donat que les dependències mai s'acabaven: uns mòduls necessitaven d'altres mòduls i així successivament. A més la funció no estava pensada per un sistema com la Nintendo DS ja que utilitzava moltes coses que aquesta a priori no disposa, com per exemple threads, consultes a fitxers de sistema com `/etc/hosts`, etcètera.

Em vaig posar en contacte amb Stephen Stair (sgstair), el creador de dswifi, la llibreria de xarxa de Nintendo DS (explicada a l'apartat 3.6), i li vaig preguntar en quina implementació de C s'havia basat per crear la seva llibreria, i en concret la funció `gethostbyname()`, molt semblant a l'altra. Em va contestar que havia creat la llibreria des de zero, i que la millor manera d'implementar `gethostbyaddr()` per la Nintendo DS seria copiant `gethostbyname()` i modificant-la.

I així ho vaig fer. Però per no copiar tot el codi de la funció creada, faré un resum dels detalls més destacables de la seva programació. Per veure una implementació completa d'aquesta funció es pot consultar l'arxiu `dswifi-src-0.3/arm9/sourcesgIP_DNS.c`, concretament les funcions `sgIP_DNS_gethostbyaddr()` i `sgIP_DNS_genquery_for_addr()`.

Generant consulta DNS

La llibreria dswifi inclou funcions per obtenir la interfície hardware de comunicació sense fil, obtenir el servidor DNS per omissió de la Nintendo DS, generar les estructures de dades necessàries, etc. És a dir, inclou totes les funcions necessàries per programar una funció de xarxa d'aquest tipus. Per tant, l'únic que havíem de fer era generar i enviar un missatge DNS amb una consulta adequada a les nostres necessitats, és a dir, un missatge DNS que consulti el hostname donada l'adreça; i a continuació rebre el missatge DNS de resposta i agafar els resultats que necessitem. Per fer això, era important que conegués bé el protocol DNS, ja que havia de seguir aquest estàndard. Per això vaig haver d'estudiar el document on es detalla el protocol [101]. A continuació s'explica amb més detall com són aquests missatges i quines són les dades que s'envien en la funció `gethostbyaddr()`.

Aquest es el format general d'un missatge de DNS:

- Header (12 bytes): Especifica el tipus de missatge
- Question (mida variable): Especifica el que es vol resoldre.
- Answer (mida variable): Especifica la resposta.
- Authority (mida variable): Especifica el nom de l'autoritat del domini
- Additional (mida variable): Informació addicional.

El nostre missatge de consulta tindrà un camp Header i un camp Question, i els altres estaran buits. El primer que emplenarem serà el camp Header:

- Identification (16 bits): És un identificador que relaciona els missatges de pregunta amb els de resposta. En el nostre cas, generem un identificador aleatori i ens el guardem, per després comprovar que és el mateix que rebem a la resposta.
- Flags (16 bits): Els més interessants són:
  - Flag QR: Indica si el paquet és de Query o de Reply. En el nostre cas el posem a 0, indicant que és de query (pregunta).
  - Flag RD: Indica si es vol resolució recursiva. En el nostre cas el posem a 1, indicant que sí volem resolució recursiva.
- Number of Questions (16 bits): Nombre d'entrades que tindrà el camp Question. Indiquem que 1, perquè en aquest missatge hi haurà una pregunta.
- Number of Answers (16 bits): Idem pel camp Answers. Indiquem que 0, perquè en aquest missatge no hi ha cap resposta.
- Number of Authorities (16 bits): Indiquem que 0
- Number of Additional (16 bits): Indiquem que 0

Per últim, ens faltarà omplir el camp Question. En el nostre cas, tindrà el següent format i contingut:

- QName (mida variable): Especifica l'adreça que es vol resoldre. L'adreça es dóna a l'inrevés i afegint `.in-addr.arpa` al final. Per exemple, per resoldre "`147.83.2.135`" es consultaria "`135.2.83.147.in-addr.arpa`". A més, s'ha de codificar de manera que s'ha de posar un enter davant de cada fragment indicant quants caràcters té el següent fragment, en comptes de punts. Es veu millor amb un exemple: "`www.google.es`" es transformaria en "`3www6google2es0`". La funció `gethostbyaddr()` ja s'encarrega de fer totes aquestes transformacions.
- QType (16 bits): Especifica el tipus de pregunta. En el nostre cas tindrà el valor 0x000C, anomenat PTR o Domain Name Pointer que significa que volem fer una resolució inversa.
- QClass (16 bits): Especifica el tipus d'adreça que es vol resoldre. En el cas de referir-se a una adreça d'Internet val 1.

Rebent resposta DNS

Després d'enviar aquest missatge, el servidor DNS resoldrà la nostra petició i ens enviarà una resposta. El nostre codi està preparat per analitzar aquesta resposta i extreure la informació que necessita. Per això, només hem d'estudiar com és un missatge de resposta del protocol DNS.

Primer, al camp Header, comprovarem el camp Identification, per veure si coincideix amb la Query que vam fer. També al Header tindrem el nombre de respostes que han arribat. Si arriben Questions o altres camps del missatge DNS els ignorem, i anem directament al camp Answer. El camp Answer està format per un o més Resource Records (RRs). El seu format és el següent:

- Name, Type, Class: El mateix que el camp Question.
- TTL (Time To Live, 32 bits): Nombre de segons que es pot guardar el RR en la cache.
- RDLength (16 bits): Mida del camp Rdata.
- Rdata (mida variable): informació que s'ha resolt, en el nostre cas, el nom que s'ha resolt.

La funció `gethostbyaddr()` finalment s'encarrega de retornar aquest camp `Rdata`.

Aquest procediment falla però per resoldre adreces privades, ja que si enviem a resoldre una adreça de la nostra xarxa privada a un servidor de DNS no la sabrà resoldre. Existeixen diferents maneres de resoldre aquest tipus d'adreces, per exemple, a una màquina UNIX, el primer que es fa quan es crida a la funció `gethostbyaddr()` és mirar el fitxer `/etc/hosts`, on hi ha una sèrie de noms i adreces IP guardades de diferents màquines de la xarxa, i potser es resol amb la informació d'aquest fitxer. En aquest projecte, per manca de temps, no s'ha implementat un sistema de resolució d'adreces privades, i depèn del servidor de DNS que tingui configurat la Nintendo DS en el moment de la execució, aquestes es podran resoldre o no.

5.2.3 Instal·lació de libxml2

Com ja hem estudiat anteriorment, un altre requisit del 2Pong és la llibreria libxml2. Aquesta llibreria tampoc havia estat portada a Nintendo DS, o no hem trobat cap signe de cap port, per tant també haurem de compilar-la nosaltres per Nintendo DS, igual que amb SDL_net.

Anem a la pàgina de la llibreria [63] per descarregar el codi font. En la seva secció de descàrregues trobarem vàries opcions per obtenir l'última versió, entre elles hi ha un servidor ftp. En el moment d'escriure aquest apartat l'última versió del codi era la 2.7.8

amb data 4 de novembre del 2010. Descarreguem el paquet amb el codi, en el meu cas, el `libxml2-2.7.8.tar.gz`. Descomprimim:

```
tar xzvf libxml2-2.7.8.tar.gz
```

El següent pas serà compilar la llibreria per Nintendo DS. Igual que quan vam compilar la llibreria SDL_net, tenim vàries opcions per fer-ho, per exemple amb scratchbox, autotools o amb un Makefile. Però en aquesta ocasió descriurem com fer-ho amb autotools, per tal de provar diferents sistemes de compilació.

Per començar, entrem al directori creat i executem la següent comanda:

```
./configure --host=arm-eabi LIBS="-lnds9 -ldswifi9" LDFLAGS=-L$DEVKITPRO/
   libnds/lib CFLAGS="-I$DEVKITPRO/libnds/include" --with-minimum --
   without-threads --prefix=$DEVKITPRO/libnds/ --bindir=$DEVKITARM/bin
   --datarootdir=$DEVKITARM/share
```

Aquesta comanda ens prepararà totes les eines d'autotools per compilar per l'entorn que nosaltres volem i amb les condicions que nosaltres volem. A continuació es descriuen les opcions que s'han editat:

- **host:** arquitectura en la que serà executat (s'usa només en compilació creuada).
- **LIBS:** llibreries que volem passar a l'enllaçador.
- **LDFLAGS:** opcions de l'enllaçador.
- **with-minimum:** volem la llibreria amb el mínim contingut.
- **without-threads:** volem la llibreria sense suport per multi-tasca, ja que la Nintendo DS no ho suporta.
- **prefix:** directori d'instal·lació.
- **bindir i datarootdir:** altres directoris d'instal·lació.

El següent pas seria compilar, però abans de compilar haurem d'arreglar alguns errors. Això dependrà òbviament de la versió de libxml2 que intentem compilar, però per la versió 2.7.8 ens trobarem amb els errors que s'indiquen a continuació.

Variable no definida

Si intentem compilar, el primer error amb el que ens trobarem serà:

```
xmllint.c: In function 'main':
xmllint.c:3353: error: 'format' undeclared (first use in this function)
```

Si analitzem el codi en aquest arxiu, veurem que la variable format està definida dins una macro, i per tant, no es podrà utilitzar si no està activat aquesta macro:

```
#ifdef LIBXML_OUTPUT_ENABLED
static int format = 0;
```

Per tant, haurem de fer que la variable `format` només es pugui utilitzar en el cas que aquesta macro hagi estat definida. Això implicarà canviar el següent codi:

```
#ifdef LIBXML_OUTPUT_ENABLED
    format = atoi(argv[i]);
#endif /* LIBXML_OUTPUT_ENABLED */
    if (format == 1) {
        noblanks++;
        xmlKeepBlanksDefault(0);
    }
```

Per aquest altre:

```
#ifdef LIBXML_OUTPUT_ENABLED
    format = atoi(argv[i]);
    if (format == 1) {
        noblanks++;
        xmlKeepBlanksDefault(0);
    }
#endif /* LIBXML_OUTPUT_ENABLED */
```

Llibreria no definida

El segon error que ens trobarem és el següent:

```
testrecurse.c:(.text+0x1450): undefined reference to 'glob'
testrecurse.c:(.text+0x17f4): undefined reference to 'globfree'
```

Que significa que aquestes funcions no estan implementades per a la Nintendo DS. Però en realitat no hi ha tal problema, perquè la llibreria s'haurà compilat. Aquestes funcions s'estan utilitzant a un test d'una col·lecció de tests que es compila després de la llibreria. Tot i així, el més correcte seria acabar de compilar. La manera més senzilla i ràpida de fer-ho és modificar el `Makefile` de manera que no agafi el test que dóna problemes, anomenat

`testrecurse`. Només hem d'editar el `Makefile` i eliminar l'aparició de `testrecurse.c` (en el meu cas a la línia 609).

Compilació, instal·lació i proves

Si volem instal·lar automàticament necessitarem haver arreglat l'error anterior, és a dir, eliminat l'aparició de `testrecurse.c` del Makefile. Llavors només haurem d'executar `make install`. Com que ja ho havíem configurat correctament, això ja ens copiarà tot el que necessitem en els directoris que hem especificat.

Si en canvi volem instal·lar a mà, haurem d'executar les següents comandes des del directori base:

```
cp .libs/libxml2.a $DEVKITPRO/libnds/lib/
mkdir $DEVKITPRO/libnds/include/libxml2
cp include/libxml $DEVKITPRO/libnds/include/libxml2/
```

Per provar que la instal·lació ha estat correcta, hi ha tota una sèrie de tests al directori arrel. Per exemple, podem compilar `testchar.c` o qualsevol altre. Però igual que hem dit abans haurem de fer modificacions per a que funcioni en Nintendo DS, per tal de compilar-lo per la plataforma o que no utilitzi els paràmetres per línia de comandes (argc i argv), entre d'altres.

5.3 Instal·lació de l'entorn de desenvolupament de PSP

En aquesta secció s'estudiarà com muntar l'entorn de desenvolupament necessari pel 2Pong per a la plataforma PSP. Començarem per introduir el procés d'alliberament de la plataforma, després explicarem com instal·lar el toolchain i el port de la llibreria SDL. Per últim veurem com compilar SDL_net i libxml2 per a PSP.

5.3.1 Alliberament de la plataforma

El firmware de PSP prohibeix que es pugui executar homebrew. Per tant per poder executar els nostres programes en una PSP haurem de canviar el firmware. Consisteix en fer un procés anomenat *downgrade*, per baixar a la versió 1.5 del firmware i després instal·lar un *Custom Firmware*, un firmware alternatiu que ens permeti saltar-nos aquesta limitació i executar homebrew. Donat que podem obtenir tota la informació en aquesta font [19], no repetirem els passos en aquesta documentació.

En el meu cas particular, vaig partir d'un firmware original 3.95, amb un model de consola PSP1004. Vaig instal·lar un Custom Firmware 5.00 M33-4.

5.3.2 Instal·lació del toolchain PSPDEV i la llibreria SDL

Per instal·lar el toolchain PSPDEV i la llibreria SDL (recordem que aquestes dues eines s'engloben dins el paquet PSPSDK), vaig seguir el tutorial [110]. Però el problema és que en el moment que jo vaig instal·lar els repositoris que indicaven els scripts estaven caiguts, pel que vaig haver de fer modificacions. Per això descriurem el procés amb brevetat:

Primer comprovem que tenim tots els paquets necessaris instal·lats. Si estem en un entorn Debian GNU/Linux o derivats ho podem fer amb el gestor de paquets APT [5]:

```
sudo apt-get install build-essential autoconf automake bison flex
   libncurses5-dev libreadline-dev libusb-dev texinfo libgmp3-dev
   libmpfr-dev subversion
```

Després editem el fitxer `~/.bashrc` i afegim aquestes línies al final amb les variables d'entorn de PSP:

```
export PSPDEV="/usr/local/pspdev"
export PSPSDK="$PSPDEV/psp/sdk"
export PATH="$PATH:$PSPDEV/bin:$PSPSDK/bin"
```

Actualitzem els canvis amb:

```
source ~/.bashrc
```

El següent pas serà descarregar-nos una còpia del codi que necessitarem. Utilitzarem el sistema de control de versions Subversion [9]. Però abans creem un directori on descarregar tot el codi i entrem en ell, per exemple anomenat "`toolchain-inst`".

El repositori oficial és:

```
svn co svn://svn.ps2dev.org/psp/trunk/pspsdk
```

Però quan jo vaig provar aquest servidor no funcionava i pel que comenta la comunitat es passa bastant temps en aquest estat. Vaig trobar un repositori alternatiu a:

```
svn co http://psp.jim.sh/svn/psp/trunk/
```

La manera més senzilla d'instal·lar és executant uns scripts que ens automatitzaran tota la instal·lació. Però si hem canviat el repositori també haurem de modificar abans aquests

scripts, ja que també han d'accedir al repositori (si no hem canviat el repositori podem saltar aquest pas).

Els scripts que haurem de modificar són els que estan ubicats als directoris:

`toolchain-inst/trunk/psplibraries/scripts/`
`toolchain-inst/trunk/psptoolchain/scripts/`

I haurem de canviar:

`svn://svn.ps2dev.org/psp/trunk/`

Per:

`http://psp.jim.sh/svn/psp/trunk/`

Per últim, executem els scripts (des dels seus corresponents directoris):

```
sudo ./toolchain-sudo.sh
sudo ./libraries-sudo.sh
```

5.3.3 SDL_net i libxml2

Les llibreries SDL_net i libxml2 no s'instal·len amb el toolchain.

SDL_net

En el cas de SDL_net podem trobar un port a la xarxa, que podem descarregar amb la comanda (utilitzant el sistema de control de versions Subversion [9]):

```
svn checkout http://coolkehon-ports-psp.googlecode.com/svn coolkehon-
   ports-psp-read-only
```

Després compilem amb la comanda `sudo make install`.

libxml2

En el cas de libxml2 no vaig trobar un port directe, però sí un tutorial per compilar-la. El tutorial està a [13] però explicaré breument el procés ja que està en francès.

Primer ens descarreguem un codi de libxml2 de [63] i el desempaquetem. Creem un fitxer "`dup.c`" al directori arrel amb el següent contingut:

```
#include <fcntl.h>
int
dup (int fd1) {
   return (fcntl (fd1, F_DUPFD, 0));
}
```

Modifiquem l'arxiu "`Makefile.am`" y afegim "`dup.c`" a les línies 36 y 46. Ara editem el fitxer "`config.sub`" i afegim el següent contingut a la línia 911 després de ps2:

```
psp)
    basic_machine=mipsallegrexel-psp
    os=-elf
    ;;
```

A continuació executem la següent comanda:

```
./configure --host=psp --disable-shared --prefix=$(psp-config --psp-
   prefix) CFLAGS="-g -O2 -G0 -Wall" LDFLAGS="-L$(psp-config --psp-
   prefix)/lib -L$(psp-config --pspsdk-path)/lib" LIBS="-lz -lc -lstdc++
    -lpsplibc -lpspuser -lpspnet_inet -lpspnet_resolver"
```

Compilem amb `make`. Sortiran errors, però són dels tests, la llibreria haurà compilat. A continuació executem `sudo make install`. També sortiran errors relacionats amb els tests, però haurà fet quasi tota la feina. Només faltarà copiar els arxius:

```
sudo cp .libs/libxml2.a /usr/local/pspdev/psp/lib/
sudo cp libxml2.la /usr/local/pspdev/psp/lib/
sudo cp libxml-2.0.pc /usr/local/pspdev/psp/lib/pkgconfig/
```

5.4 Scratchbox2

En aquest punt (si el lector ha avançat seqüencialment) hem vist com instal·lar un entorn de desenvolupament per cada màquina, Nintendo DS i PSP. Però ja hem dit diverses vegades que experimentaríem amb altres tipus d'entorns de desenvolupament de compilació creuada. El nostre propòsit inicial era provar entorns diferents per estudiar-los i aprofitar-nos dels seus avantatges. A la secció 3.3 vam dir que experimentaríem amb Scratchbox2, i encara que finalment no s'ha utilitzat per un tema de comoditat (personalment m'agradaven més els Makefile que proveeixen els toolchain), en aquesta secció detallarem com instal·lar-lo i usar-lo.

Hem obtingut la informació principalment d'aquest tutorial [55], encara que en ell s'usa un toolchain per ARM genèric i nosaltres el volem vincular amb el nostre toolchain, pel que

variarem una mica el procediment. Instal·larem Scratchbox2 en sí i QEmu, un emulador d'ARM.

Paquets necessaris

Per a la instal·lació completa necessitarem els següents paquets, la majoria d'ells ja els hem instal·lat durant la instal·lació del toolchain: autoconf, autogen, automake, autotools-dev, binutils, fakeroot, git-core, g++, make, sbrsh.

Els paquets específics per a la instal·lació de QEmu són: gcc-3.4, libsdl1.2-dev, subversion.

Si estem en un entorn Debian GNU/Linux o derivats podem usar el gestor de paquets APT (Advanced Packaging Tool) [5] per assegurar-nos de que els tenim tots instal·lats. Cal dir que si ens trobem sobre una altra distribució, haurem d'usar un altre mètode de descàrrega.

```
sudo apt-get install autoconf autogen automake autotools-dev binutils
   fakeroot gcc-3.4 git-core g++ libsdl1.2-dev make sbrsh subversion
```

Crear directoris

Abans de la instal·lació hem de crear la següent estructura de directoris:

- sbox2
  - bin (Conté tots els binaris necessaris per a l'entorn)
    - qemu (Binaris de QEmu)
  - rootfs (Conté els sistemes de fitxers arrel per a diferents arquitectures)
    - nds (Sistema de fitxers arrel per a Nintendo DS)
  - src (Codi font per a Scratchbox2)

Per crear els directoris executem:

```
mkdir -p ~/sbox2 ~/sbox2/bin/qemu ~/sbox2/rootfs/nds ~/sbox2/src
```

Crear el sistema de fitxers arrel

Un pre-requisit per Scratchbox2 és crear un sistema de fitxers arrel. En dispositius com iPhone, podem copiar el sistema de fitxers del dispositiu. Scratchbox2 està originalment pensat per sistemes amb Debian, i per això demana un sistema de fitxers. En el cas de

Nintendo DS, no tenim sistema operatiu, i per tant no tenim un sistema de fitxers arrel per copiar.

Usar toolchain

Per a que Scratchbox2 pugui usar el nostre toolchain primer hem d'incloure la localització del nostre compilador creuat al PATH del sistema. Per a que el canvi sigui permanent editem l'arxiu `.bashrc` que es troba a la nostra home i hi afegim la següent línia. Si no tenim l'arxiu el creem. En aquest cas concret, com que estem configurant scratchbox per usar el toolchain de Nintendo DS, fem ús de la variable d'entorn `$DEVKITARM`, que vam crear quan instal·làvem el toolchain.

```
vim $HOME/.bashrc
export PATH=$PATH:$DEVKITARM/bin
```

Tanquem la consola on hem editat l'arxiu i n'obrim una de nova per tal que s'apliqui el canvi.

Scratchbox2

Per a instal·lar Scratchbox2 ens descarreguem la darrera versió del codi font i la compilem usant les comandes:

```
cd ~/sbox2/src
git clone git://gitorious.org/scratchbox2/scratchbox2.git sbox2
cd sbox2
git checkout 2.1 -b devel_env
./autogen.sh
./configure --prefix=$HOME/sbox2/bin/scratchbox
make install
```

Per finalitzar hem d'afegir algunes opcions de configuració: necessitem definir l'arquitectura del sistema host. Per fer-ho, s'ha d'editar el fitxer:

`~/sbox2/bin/scratchbox/bin/sb2-build-libtool`

I buscar una línia que executa el script de configuració i afegir-hi el sistema host de la forma `--host=i386`. En el nostre cas la línia queda de la següent manera:

```
./configure --prefix=$HOME/.scratchbox2/$TARGET --build=$(uname -m)-
   unknown-linux-gnu --host=i386
```

QEmu

Una part de l'entorn és l'emulador, usarem QEmu. Si ja disposem de QEmu al nostre sistema l'haurem d'eliminar usant la comanda:

```
sudo apt-get --purge remove qemu
```

Usem Git per descarregar el codi font:

```
cd ~/sbox2/src
git clone git://git.savannah.nongnu.org/qemu.git
cd qemu
git checkout v0.13.0-rc1 -b devel_env
```

Ara ja podem compilar el codi de QEmu:

```
./configure --prefix=$HOME/sbox2/bin/qemu
make
make install
```

Ús

Abans de poder usar Scratchbox2 ens situem al directori on hem creat el sistema de fitxers arrel i inicialitzem allà Scratchbox2 indicant quin compilador creuat usarem, en el nostre cas l'`arm-eabi-gcc`. Això farà que a l'iniciar Scratchbox2 es crei un entorn chroot en aquest directori i quan usem la comanda `gcc` aquesta apunti a `$DEVKITARM/bin/arm-eabi-gcc`.

```
cd ~/sbox2/rootfs/nds
sb2-init nds $DEVKITARM/bin/arm-eabi-gcc
```

Per usar Scratchbox2 hem d'usar la comanda `sb2`. L'executem a una consola i veurem que l'aspecte del prompt canvia:

```
user@laptop:~/sbox2$ sb2
[SB2 simple nintendo-ds] user@laptop nds $
```

```
echo 'int main() { return 0; }' > test.c
gcc test.c
file a.out
a.out: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically
   linked, not stripped
```

Tenim un altre exemple del seu ús a la figura B.4 als apèndixs.

Un cop ja hem provat Scratchbox2 ja hem acomplert l'objectiu d'experimentar amb altres entorns de compilació creuada. Tot i que ens ofereix avantatges com un entorn segur de desenvolupament, ens sembla menys còmode que altres mètodes de compilació, pel que no l'hem usat normalment. Però sempre és bo haver-lo provat, per si el volem utilitzar en algun moment.

5.5 Adaptació i compilació del joc

Ara que ja tenim tot l'entorn de desenvolupament necessari instal·lat, ja estem preparats per portar el joc. Però abans haurem de preparar el codi del joc per a que pugui ser executat en les diferents plataformes.

En aquesta secció s'expliquen els canvis que s'han dut a terme per adaptar a les plataformes Nintendo DS i PSP un joc que originalment estava pensat per ser executat en un PC. Veurem que s'han hagut de fer canvis com adaptar a la resolució de pantalla, o adaptar els controls. També s'expliquen els canvis generals fets en el codi per totes les versions. Per últim, veurem com compilar aquest codi per els dos dispositius.

5.5.1 Compilació condicional

A vegades necessitarem separar el codi específic de Nintendo DS i PSP de la resta de plataformes. Això passarà per exemple si utilitzem una funció d'inicialització que només s'ha d'executar en Nintendo DS i que no existeix per PC. Si intentéssim compilar per PC donaria error de compilació, donat que la funció no existeix per aquesta plataforma. En aquesta secció veurem un exemple amb aquesta plataforma però s'aplicaria igual per tota la resta.

Per fer-ho utilitzarem compilació condicional gràcies a les directives de preprocés de C. Utilitzarem una macro que només està definida al Makefile de Nintendo DS amb el nom de `__NDS__`, i com que només estarà definida per aquesta plataforma, amb ella podrem controlar l'accés a determinades parts del codi.

Amb el següent exemple s'entendrà millor. Primer, es comprova si la macro `__NDS__` està definida. Si és així (significa que estem compilant per Nintendo DS) es definirà l'alçada de la finestra com 192, sinó (s'està compilant per a una altra plataforma) es definirà com 480.

```
#ifdef __NDS__
        #define WINDOW_HEIGHT 192
#else
        #define WINDOW_HEIGHT 480
#endif
```

5.5.2 Canvis en el joc per a la conversió a Nintendo DS

A l'hora d'adaptar el codi original a la consola Nintendo DS (o a qualsevol altra plataforma), el cas ideal era fer el mínim de canvis possibles al codi original, però van sortir problemes que ens ho van impedir. Per exemple, no es visualitzava bé el joc, o no es podia controlar tal i com estava implementat. Per tant no ens quedava més remei que fer modificacions. En aquesta secció veurem quins van ser i com es van solucionar aquests problemes. Veurem que hi ha tres canvis més extensos per la seva importància, i a continuació una sèrie de canvis més petits.

Resolució de pantalla

La resolució original del 2Pong és 640x480 píxels, però la Nintendo DS té una pantalla (de fet en té dues però nosaltres només n'usarem una) de 256x192 píxels. Les dues resolucions són 4:3, però una es 2,5 vegades més petita que l'altra.

La llibreria SDL està preparada per això, i escala sola la imatge al tamany de la pantalla de Nintendo DS, és a dir, encara que el joc declari una pantalla de 640x480 píxels:

```
screen = SDL_SetVideoMode (640, 480, BITS_PER_PIXEL ,SDL_SWSURFACE);
```

El contingut s'escalarà sol. Això hauria de funcionar sense problemes, de fet per escalar resolucions de 320x240 no dóna problemes, i alguns casos de 640x480 també funcionen. No obstant, per una mala gestió dels recursos de la màquina per part del port de SDL, no hi ha suficient memòria i aquest escalat no funcionava bé pel 2Pong. Sortia una franja negra al mig de la pantalla en la que no es podia visualitzar res. Aquest mateix resultat l'he experimentat en altres exemples de Nintendo DS en els que he intentat escalar una pantalla de 640x480. Per tant, no podrem utilitzar aquest auto-escalat.

Una solució a aquest problema és adaptar tota la resolució del joc a 256x192 píxels per a la versió de Nintendo DS. Només haurem de canviar l'anterior crida per:

```
screen = SDL_SetVideoMode (256, 192, BITS_PER_PIXEL ,SDL_SWSURFACE);
```

Però això implica alguns canvis de diferent índole. Per una banda, s'hauran d'adaptar els sprites que s'utilitzen. El joc utilitza uns gràfics (unes imatges bmp) que estan preparats per a la resolució original del joc. Si modifiquem la resolució de la pantalla, haurem de reduir aquestes imatges en proporció, sinó el joc quedarà desproporcionat respecte l'original.

Però aquest canvi té un inconvenient que apareix quan es juga una partida en xarxa entre dos dispositius amb resolució diferent. Aquest problema és degut a com està implementat el joc, ja que utilitza coordenades de dispositiu i no coordenades de món. Aquestes coordenades es guarden en píxels, i quan enviem unes dades ho estarem fent en coordenades

referents al nostre dispositiu i no coincidiran amb la mateixa posició a l'altre dispositiu.

Ho veurem millor amb un exemple: imaginem una bola que es troba al punt mig de la pantalla, tindria coordenades (320,240) en un PC, i (128,96) en una Nintendo DS. Així, si el jugador amb la Nintendo DS envia les seves coordenades al jugador amb el PC, aquest pintarà la bola a la posició (128,96), que segons la seva resolució i el seu sistema de coordenades no serà el punt mig de la pantalla. No estaran pintant el mateix escenari.

Per tant les dades que s'envien per la xarxa han d'estar normalitzades; tots els sistemes han de parlar el mateix llenguatge o sinó no s'entendran. El joc pot funcionar a un sistema de coordenades diferent en cada dispositiu, però a l'hora d'enviar les dades per la xarxa les normalitzarem a alguna convenció acceptada per totes les versions.

S'ha escollit la següent solució: adaptar les dades que s'envien i es reben per la xarxa segons quina sigui la resolució de la plataforma en la que estem. Les dades que s'han de tractar són la posició i velocitat de les boles i la posició de les pales.

Per acabar d'entendre la solució proposada, estudiarem una porció del codi implementat. Es defineixen unes constants que depenen de la plataforma, i que indiquen en quina escala s'ha reduït la pantalla respecte l'original:

```
#ifdef __NDS__ // plataforma = Nintendo DS
#define FACTOR_REDUCCIO_WINDOW 2.5
#else  // altres plataformes
#define FACTOR_REDUCCIO_WINDOW 1
#endif
```

Aquestes constants s'utilitzen per escalar les dades a l'hora d'enviar-les i rebre-les per la xarxa:

```
out->data[0] = paddles[1].GetY() * FACTOR_REDUCCIO_WINDOW;
```

```
paddles[1].yset(in->data[0]/FACTOR_REDUCCIO_WINDOW);
```

Per exemple en una comunicació entre una Nintendo DS i un PC, la Nintendo DS multiplicarà les dades que envia per un factor de 2.5, i el PC les dividirà per 1, així que l'escalat serà correcte.

Controls

El codi original del 2Pong utilitza els següents mètodes de captura de l'entrada: el ratolí per moure's pel menú, i el teclat per controlar el joc pròpiament dit. El problema és que cap dels dos *inputs* funcionaven directament a la Nintendo DS. En el cas del teclat és evident, ja que la Nintendo DS no disposa d'un teclat físic amb el que controlar el joc. En

el cas del ratolí seria possible portar la funcionalitat a la Nintendo DS gràcies a la seva pantalla tàctil, i ho explicarem en aquesta secció. Però en tot cas, novament s'havien de fer canvis en el codi del 2Pong per adaptar els controls que no funcionessin, i que el joc fos controlable per aquesta plataforma.

En el cas del ratolí, no ha estat portat a Nintendo DS, ho diu el mateix autor del port al seu lloc web [49], quan diu "to do: mouse/keyboard support" (per fer: afegir suport mouse/teclat), i és que com veurem al següent apartat (teclat per pantalla), hi ha un problema amb la pantalla tàctil que el fa difícil de portar. Tot i així no seria impossible, ja que jo he aconseguit utilitzar la interfície tàctil juntament amb SDL.

Per tant, a l'hora de modificar els controls, si no podem comptar amb SDL_mouse, el més correcte hagués estat afegir la interfície SDL_Joystick, ja que és la solució més portable. Així el codi que afegim al 2Pong servirà per la Nintendo DS, que té portada la interfície SDL_Joystick; però també servirà per si algú vol jugar en PC amb un joystick, o també per si en un futur es fa un port per a una altra plataforma que tingui joystick. Seria una solució més multi-plataforma.

Com ja hem dit, la interfície SDL_Joystick ha estat portada a Nintendo DS. El problema és que pels experiments realitzats durant aquest projecte podem afirmar que SDL_joystick no respon de manera tan acurada com les funcions d'input de libnds: SDL_Joystick freqüentment es salta alguns events d'entrada, i és molest jugar així. És per aquest motiu que finalment vam decidir utilitzar les funcions de libnds, tot i sabent que no és tan correcte, perquè és una solució depenent de la plataforma, però dóna una experiència de joc més satisfactòria. Per provar el port de SDL_Joystick en funcionament, es pot compilar l'exemple anomenat `sdl_wingman` el qual forma part del paquet dels exemples del port de SDL de GPF, que vam descarregar durant la instal·lació d'aquest (apartat 5.1.3). Així es podrà provar en primera persona que moltes vegades no respon correctament als botons que estem prement.

Per utilitzar els controls de libnds farem així:

```
# ifdef __NDS__ // Si estem a la plataforma Nintendo DS
    scanKeys(); // Actualitzar estat dels botons
    if (keysHeld() & KEY_A) // Si esta premut el boto A
        [...]
# else
    // Programacio dels controls per PC
# endif
```

Teclat per pantalla: La necessitat d'un programa ARM7

No obstant encara queda una funcionalitat referent als controls per substituir: el teclat per pantalla. Quan client o servidor han d'escriure adreces o ports, necessiten algun mitjà per escriure i en Nintendo DS no comptem amb un teclat físic com en un PC. Per tant

hauríem de mostrar un teclat per pantalla, i que l'usuari pogués manipular el teclat amb els botons o amb la pantalla tàctil. libnds ens permet mostrar un teclat per pantalla que porta la Nintendo DS de sèrie guardat al firmware de la consola, i ens aporta tota una sèrie de funcions per manipular-lo a través de la pantalla tàctil [59].

El problema és que em vaig adonar que en inicialitzar SDL deixava de funcionar el teclat per pantalla de libnds. Després de fer algunes proves, vaig descobrir que també estaven fallant les funcions de libnds per llegir la pantalla tàctil, els botons X i Y i d'altres funcionalitats. Si ens hi fixem totes les funcionalitats que estan fallant estan controlades pel processador ARM7: tant la pantalla tàctil com els botons X i Y estan controlades per aquest processador esclau. I sense pantalla tàctil no podrem usar el teclat per pantalla de libnds, que es controla a través d'aquesta interfície.

Existeix una regió de memòria per a la comunicació entre els dos processadors, en la que s'intercanvien per exemple els valors obtinguts de llegir la pantalla tàctil (l'ARM7 la llegeix i li passa les dades a l'ARM9). Per algun motiu, la inicialització de SDL corromp aquesta regió de memòria. Per tant, si utilitzem SDL perdrem la comunicació entre els processadors i no podrem utilitzar la pantalla tàctil.

Però vaig trobar una solució per poder seguir utilitzant SDL i la pantalla tàctil alhora. libnds oferia una estructura anomenada `IPC` (Inter Process Control, del tipus `TransferRegion`) amb la que es podia controlar aquesta porció de memòria. Si fem un programa ARM7 personalitzat en comptes d'usar el que s'enllaça per omissió, podrem tornar a controlar aquesta porció de memòria i tornar a enviar les dades col·lectades per l'ARM7. L'únic problema és que aquesta estructura es considera obsoleta i no es recomana el seu ús.

Aquest programa ARM7 que utilitzem, com que substituirà el programa ARM7 per omissió, ha d'encarregar-se de fer totes les tasques habituals que faria aquest processador, que ve a ser la lectura de l'entrada, la comunicació sense fil i l'àudio. Vaig haver de buscar molts programes i fer moltes proves per obtenir un programa que permetés accedir a totes les funcionalitats. Per exemple, el port de SDL inclou un programa ARM7 d'exemple que usa la estructura `IPC`, i per tant funcionen els controls, però no té implementada la Wi-Fi. Podem veure el codi final als fonts del 2Pong, a la figura B.3 dels apèndixs.

Per últim, dir que la solució més correcta hagués estat arreglar el port de SDL per tal que no corrompés aquesta porció de memòria i pogués funcionar amb qualsevol programa ARM7, incloent els programes per omissió.

Error de compilació: Paraula reservada

Si provem de compilar el 2Pong, ens trobarem amb un error de compilació que només apareix en la compilació de Nintendo DS: el codi original del joc defineix una variable anomenada `Keyboard`, que malauradament coincideix amb una paraula clau de libnds. Per tant, només per aquesta versió haurem de canviar el nom de la variable `Keyboard` al codi

del 2Pong, i modificar cada aparició d'aquesta variable al codi. La variable està definida a `defs.h` i s'utilitza a `game.cpp`, `main.cpp` i `menu.cpp`.

Inicialitzacions

Haurem de cridar a les funcions de libnds per iniciar consola:

```
consoleDemoInit();
```

Per poder mostrar missatges per pantalla. I tambe per iniciar FAT:

```
fatInitDefault();
```

Per poder accedir al sistema d'arxius.

Connectar a la xarxa

S'ha hagut de creat una funció específica de Nintendo DS per connectar a la xarxa. Les altres plataformes no necessiten cridar una funció per connectar-se, per exemple el PC es suposa que ja està connectat quan executa el 2Pong. Aquesta funció, entre altres tasques, s'encarrega de cridar a:

```
Wifi_InitDefault(WFC_CONNECT);
```

Que s'encarrega d'inicialitzar la Wi-Fi i connectar a una de les xarxes que es tinguin configurades per omissió a la consola. A més, es mostren alguns missatges per a l'usuari, com per exemple la IP amb la que s'ha connectat.

Funció de pintat

La funció `SDL_UpdateRects()` que utilitza el joc no ha estat portada a la Nintendo DS. Podem comprovar-ho mirant l'arxiu `SDL_ndsvideo.c` del port, on veurem que la funció `NDS_UpdateRects()` està buida. Aquesta funció serveix per actualitzar només les parts que li indiquem de la pantalla mitjançant una llista de rectangles. En el seu lloc cridarem a la funció `SDL_UpdateRect(screen, 0, 0, 0, 0);` que actualitza tota la pantalla i és més lenta, però sí que està portada.

Capçaleres

Per últim, haurem d'afegir els arxius de capçalera específics de Nintendo DS:

```
#ifdef __NDS__
#include <nds.h>      // llibreria libnds
#include <fat.h>      // llibreria libfat, per acces a disc
#include <dswifi9.h> // llibreria dswifi, xarxa del processador arm9
#endif
```

5.5.3 Canvis en el joc per a la conversió a PSP

Igual que vam dir en la secció de Nintendo DS, l'ideal és fer el mínim de canvis en el codi original, però haurem de fer canvis per connectar a la xarxa o mostrar un teclat per pantalla. A continuació es mostren els més rellevants.

Teclat virtual: diàlegs natius de PSP

Per el moment en què s'ha d'introduir una adreça o un port es necessitava un teclat. A l'igual que altres plataformes, la PSP porta un teclat per pantalla integrat en el firmware. Però el problema és que aquest teclat utilitza la llibreria gràfica pròpia de PSP, anomenada GU. Llavors per mostrar el teclat, canviàvem de llibreria gràfica i es perdia la visualització amb SDL. Es van provar altres alternatives, com el "teclat de Danzeff" [20], un teclat creat per PSP que pot ser usat amb SDL, però no es va poder integrar en el joc.

Finalment, vaig consultar a una comunitat de PSP [86] i em van donar una solució: sí que es podia utilitzar el teclat per pantalla del firmware, si s'integraven de manera correcta les dues interfícies gràfiques, SDL i GU. És a dir, cada cop que vulguem cridar a un diàleg natiu de PSP, com per exemple el teclat, seguirem el següent procés:

- Finalitzar SDL
- Inicialitzar GU
- Mostrar el diàleg de PSP i fer el que necessitem. En aquest cas: mostrar el teclat, deixar que l'usuari escrigui i retornar el valor escrit per l'usuari.
- Finalitzar GU
- Inicialitzar SDL

Aquest mètode també s'ha utilitzat per mostrar un diàleg natiu de PSP que mostra les xarxes sense fil que hagin estat prèviament configurades i permet escollir a quina et vols connectar. Fins i tot permet configurar noves xarxes.

Connectar a la xarxa

Per connectar a la xarxa amb PSP també havíem de posar codi específic de la plataforma.

A l'hora de programar per una consola, una bona font d'informació és mirar els exemples que acompanyen al toolchain. Amb aquests exemples es pot aprendre a fer moltes coses.

Quan vaig voler connectar a la xarxa amb la PSP, vaig consultar aquesta font, però el problema és que quan provava exemples de la PSP, molts no em funcionaven: no arrancaven la execució i em donaven un codi d'error de sistema. Aquests codis estan recopilats a [6], una font molt útil, ja que la consola només et dona el codi d'error però no pots saber què significa.

En concret quan intentava compilar l'exemple rebia l'error 0x8002013a, que segons la font significa "library is not yet linked" (la llibreria encara no ha estat enllaçada).

Vaig descobrir que el codi de PSP és diferent segons les versions del firmware pel que estiguem programant. Els exemples que s'inclouen al toolchain estan preparats per ser executats amb una PSP amb un firmware 1.5, però si volem executar amb un altre, en aquest cas un CF, haurem de fer algunes modificacions. En aquest cas, faltaven les següents inicialitzacions:

```
sceUtilityLoadModule(PSP_MODULE_NET_COMMON);
sceUtilityLoadModule(PSP_MODULE_NET_INET);
```

Que serveixen per inicialitzar les llibreries de xarxa.

Problemes de connexió entre client i servidor

El problema és que després d'usar el teclat per pantalla de PSP que hem explicat abans, el client i el servidor no podien establir una connexió. L'error està relacionat amb les funcions `SDLNet_ResolveHost` i `showOSKDialog`, així que les introduirem:

La funció de SDL_net per resoldre un host és:

```
int SDLNet_ResolveHost(IPaddress *address, char *host, Uint16 port);
```

Que donada una IP i un port ens resoldrà el host. Un exemple d'utilització és:

```
IPaddress address;
SDLNet_ResolveHost(&address, "192.168.1.34", 5555);
```

La funció que hem creat per mostrar i llegir del teclat de PSP és:

```
char * showOSKDialog();
```

Que retorna la cadena escrita per l'usuari.

El problema és que si utilitzem SDLNet_ResolveHost per resoldre l'adreça que ha introduït l'usuari:

```
SDLNet_ResolveHost(... , showOSKDialog() , ...);
```

La funció SDLNet_ResolveHost no aconsegueix resoldre la IP que li demanem i no es connecten el client i el servidor. Però sabem que showOSKDialog() està funcionant correctament, perquè si fem la prova:

```
char res[128] = showOSKDialog();
printf("%s\n",res);
```

Ens mostra un resultat correcte. Per tant, l'única explicació és que al sortir de la funció `showOSKDialog` i entrar a la funció `SDLNet_ResolveHost` ja no podem accedir a la posició de memòria a la que apunta el resultat de `showOSKDialog`:

```
char * showOSKDialog() {
    char resultat [128];
    ...
    return resultat;
}
```

La solució implementada consisteix en fer una còpia en memòria del resultat de showOSKDialog:

```
char aux [128];
SDLNet_ResolveHost(... , strcpy(aux,showOSKDialog()) , ...);
```

I així si que funciona.

Mostrar adreça IP servidor

Per últim, s'ha afegit codi per mostrar per pantalla l'adreça IP del servidor. Ja que sinó no podrem saber quina adreça li ha assignat la xarxa i el client no podrà connectar-se.

Per obtenir l'adreça IP que ens han assignat usarem el següent codi:

```
char myIPaddr[32];
union SceNetApctlInfo apctlInfo;
strcpy(myIPaddr, "");
if (sceNetApctlGetInfo(PSP_NET_APCTL_INFO_IP, &apctlInfo)<0){
        strcpy(myIPaddr, "unknown IP address");
}else{
        strcpy(myIPaddr, apctlInfo.ip);
}
```

Controls

Per adaptar el joc a la plataforma PSP també s'han d'adaptar els controls del joc. Es podria haver utilitzat SDL_Joystick, però es va trobar una solució millor a [61]. Diem que és millor solució perquè implica menys modificacions en el codi del joc.

Aquesta solució consisteix en realitzar un mapeig directe de les tecles que utilitza el joc pels botons de PSP, canviant la implementació de la classe Keyboard (utilitzant la API específica de PSPSDK).

Tot i així haurem de fer algunes adaptacions sobre aquesta implementació, com per exemple canviar algunes tecles que defineix per les que utilitza el 2Pong:

```
configured_keys[SDLK_ESCAPE] = PSP_CTRL_CROSS;
configured_keys[SDLK_SPACE] = PSP_CTRL_CIRCLE;
configured_keys[SDLK_p] = PSP_CTRL_START;
```

Per a que aquesta implementació pugui servir també pel menú haurem de canviar el control per ratolí per control per teclat en la versió de PSP.

Capçaleres

Per últim, haurem d'afegir els següents arxius de capçalera específics de PSP:

```
#ifdef __PSP__
#include <pspgu.h>        // Interficie grafica propia de PSP
#include <pspdisplay.h>   // Utilitats interficie grafica
#include <psputility.h>   // Necessaria pel teclat
#include <pspnet.h>       // Xarxa
#include <pspnet_inet.h>  // Xarxa
#include <pspnet_apctl.h> // Xarxa, utilitats per Access Points
#endif
```

5.5.4 Errors en el codi del 2Pong

Al analitzar el codi del joc que vam triar vam descobrir que tenia errors que afectaven tant a la compilació com a la usabilitat. Cal remarcar que aquests errors són comuns a totes les versions, i no només específics de les versions de Nintendo DS o PSP.

Errors de compilació

Obtindrem com a mínim dos errors si intentem compilar el 2Pong. En primer lloc ens trobem aquest error:

```
game.h:6:27: error: SDL/SDL_mixer.h: No such file or directory
```

Això és perquè tenim includes d'aquesta forma:

```
#include <SDL/SDL_mixer.h>
```

Aquesta línia ens donarà error si la nostra estructura de fitxers no es correspon amb la indicada. Seria més correcte:

```
#include <SDL_mixer.h>
```

I serà a l'hora de compilar quan ens encarregarem d'indicar la ruta cap a aquest header.

L'altre missatge d'error és el següent:

```
defs.h:228: error: extra qualification 'defines::' on member 'UpperBound'
```

Aquest error, comunment anomenat *extra qualification* [15], ve d'aquest codi:

```
class defines
{
public:
        rectangle defines::UpperBound();
    [...]
};
```

En aquest tros de codi s'està definint la funció `UpperBound` dintre de la classe `defines`. Amb `defines::` estem indicant que aquesta funció pertany a la classe `defines`, cosa que és redundant. Aquesta redundància dóna error a partir de la versió 4 del compilador GCC però estava acceptada en versions anteriors.

Per tant, la solució és treure el qualificador extra:

```
rectangle UpperBound();
```

Error de teclat

En el mode de joc en xarxa, el client ha d'escriure l'adreça i el port del servidor en el format `adreça:port`. El problema és que no es captura correctament l'event de teclat que indica que s'han premut els dos punts.

El codi erroni és el següent:

```
if (event.key.keysym.sym == SDLK_SEMICOLON && (KMOD_RSHIFT || KMOD_LSHIFT
   )) {
    str[len]=58;
        len++;
}
```

La condició de l'if és errònia per dos motius. Primer, està programat pensant només en un teclat americà, on els dos punts comparteixen tecla amb el punt i coma (semicolon). Una solució ràpida seria adaptar el codi al teclat espanyol, però no seria la més correcta, s'hauria d'implementar una solució que no fos depenent del teclat que s'utilitza. SDL inclou suport per a teclats internacionals. Permet traduir events de teclat i posar els equivalents unicode a la variable `event.key.keysym.unicode`. Aquesta funcionalitat ha de ser activada amb `SDL_EnableUNICODE(1)`.

En segon lloc no està ben implementada la captura de la tecla Shift, ja que hi escriu la constant de la tecla Shift, però no la funció per llegir el seu estat, que és `SDL_GetModState()` i per tant no està fent res. Si implementem la solució proposada no ens caldrà mirar l'estat de Shift.

Fent els dos canvis el codi quedaria així:

```
SDL_EnableUNICODE(1);
[...]
else if ( str_cmp(event.key.keysym.unicode, ':') == 0 ) {
    str[len]=58;
    len++;
}
```

Paquet de mida insuficient

Durant el bucle principal del joc, si estem en mode multi-jugador, el client envia les seves dades contínuament al servidor. És un paquet de 3 bytes, i es reserva aquest espai quan es declara:

```
out=SDLNet_AllocPacket(3);
```

Però quan es preparen les dades per enviar, diu que és un paquet de 4 bytes, encara que finalment només se n'envien 3:

```
out->len=4;
out->data[0]=paddles[i].GetY()/3;
out->data[1]=abs(paddles[i].GetVelocity().y);
out->data[2]=sign(paddles[i].GetVelocity().y);
```

Això no provoca cap problema a simple vista en el joc de PC, però és un error. I en el cas concret de la Nintendo DS, fa que el joc no funcioni (bloqueja l'aplicació). És un error molt fàcil de solucionar: només hem d'enviar la mida correcta del paquet:

```
out->len=3;
```

Reserva de memòria insuficient

Per tal de connectar-se al servidor, el client ha d'escriure un hostname o adreça. En algunes ocasions l'algorisme falla en guardar aquesta informació en memòria. Això és degut a que no es reserva suficient espai de memòria. Ho veurem en el següent fragment de codi:

```
host = (char *) calloc(len,sizeof(char));
for (i=0;i<len;i++)
    host[i]=str[i];
```

Com podem veure, l'algorisme copia el nom del host des d'un altra cadena anomenada `str`. Però primer reserva espai en memòria. El problema és que reserva `len` posicions, que és el nombre de caràcters que té la cadena `str`, i hauria de reservar una posició més, `len+1`, per guardar espai pel caràcter de final de cadena `'\0'`. Així doncs, la solució serà reservar un espai més de memòria:

```
host = (char *) calloc(len+1,sizeof(char));
```

Port no autoritzat

En el mode multi-jugador, tant el client com el servidor han d'escriure el port de comunicació que volen usar. Si en qualsevol cas es posa un port més petit de 1024 l'aplicació avisa d'un error de segmentació i es tanca.

Els ports amb número inferior a 1024 són els well-known ports [95], aquests ports estan reservats per als protocols de xarxa del sistema. Només es poden usar per establir connexions entrants si tenim privilegis de xarxa.

En concret, l'error està localitzat en el moment d'enllaçar el socket al port escollit:

```
*sock=SDLNet_UDP_Open(port);
if(!(*sock)) {
    printf("ERROR SDLNet_UDP_Open: %s.\n", SDLNet_GetError());
}
```

L'anterior codi mostraria el missatge per consola:

```
ERROR SDLNet_UDP_Open: Couldn't bind to local port.
```

Una solució a aquest error seria no permetre a l'usuari introduir un port inferior a 1024. O bé que qualsevol port inferior a 1024 es transformés automàticament en 1024, que és la solució que s'ha implementat, donada la seva simplicitat.

```
port = (port<1024 ? 1024 : port);
```

Encara que creiem que la solució ideal seria que l'aplicació escollís el port mes adient automàticament, per evitar que l'usuari l'hagi d'escriure, ja que no té perquè conèixer aquesta limitació. Es podria permetre el canvi d'aquest port mitjançant un menú d'opcions.

Un altre problema de ports: dos ports iguals

De manera similar a l'apartat anterior si el port del servidor i el del client coincideixen també dóna un error de segmentació. Aquest error ocorre a la màquina del client, que és el segon en connectar-se. L'error es dóna en el mateix punt que abans, i ens dóna el mateix missatge:

```
ERROR SDLNet_UDP_Open: Couldn't bind to local port.
```

Això és degut a que el port de connexió ja està utilitzat pel servidor i no es pot enllaçar un altre socket al mateix port. La solució seria no deixar que el client escrivís el mateix port per al client que pel servidor.

El joc no acaba per al client

En acabar el joc normalment surt un missatge durant uns segons amb la puntuació final i després es torna al menú principal. En el mode multi-jugador això passa per al servidor,

però no per al client. Aquest no rep el missatge de final de partida, i no torna al menú principal, pel que no pot seguir jugant.

L'error és degut a que no es comprova l'estat del joc per al client. Hi ha una funció `CheckState` que comprova l'estat de la partida, i si aquesta ha acabat (algun jugador ha arribat a 5 punts), fa totes les gestions per sortir del joc (mostrar el missatge de puntuació, tornar al menú principal). En aquest tros de codi és quan es crida a la funció `CheckState`:

```
if (net < 2) {
    state=CheckState(game,balls,paddles,def);
    [...]
}
```

Com podem veure la crida es fa si la variable `net` és inferior a 2. Però en el cas de ser client aquesta variable té valor 2 i per tant no passa la condició.

Una solució seria comprovar l'estat del joc també quan `net` té per valor 2. Encara que la solució més correcta és que el servidor li enviï l'estat del joc al client un cop ha finalitzat.

Errors de visualització al client

En el mode multi-jugador, abans de començar a jugar cada bola, es segueix un protocol de sincronització. Tant a la pantalla del servidor com a la del client surt un missatge dient "Press Space to begin" (Prem la tecla espai per començar). El joc no comença fins que el servidor no prem la tecla Espai. Això comporta diferents errors. En primer lloc, donat que el client no ha de prémer cap tecla, se li hauria de mostrar un altre missatge, com ara "Esperant a l'altre jugador".

En segon lloc el missatge de "Press Space to begin" no s'esborra de la pantalla del client. Per un defecte en el codi aquest no rep l'ordre de refrescar la pantalla i el missatge es mostra permanentment. Després, quan la bola va passant per sobre el missatge el va esborrant i provoca un efecte molt lleig (veure figura 5.2).

El problema és que no es fa una sincronització correcta. Si el client no està en mig d'un joc està esperant rebre dades de xarxa, i quan rep qualsevol dada de xarxa comença el joc.

La solució serà la següent: reiniciar la pantalla del client quan comença el joc, tal i com es fa amb el servidor. I només fer-ho una vegada per evitar l'efecte de parpelleig o *blinking*.

```
estat_ant = estat_nou;
estat_nou = game->GetBegin();
if (estat_nou==1 && estat_ant==0){
        InitBackground(screen);
        SDL_Flip(screen);
}
```

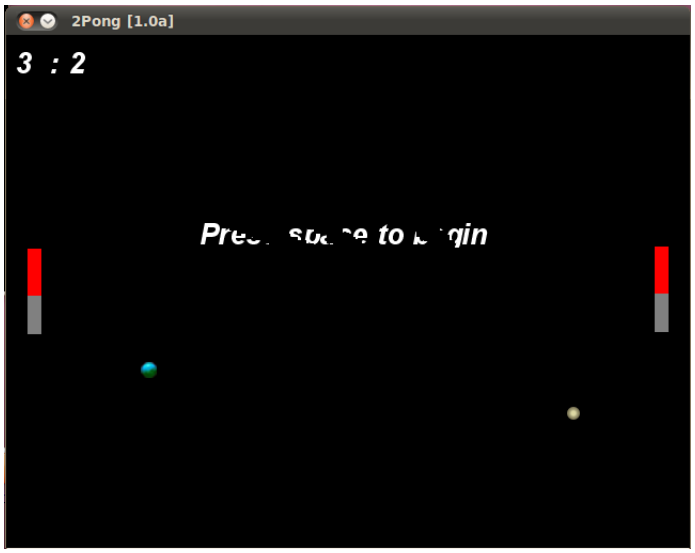

Figura 5.2: Error de visualització: El missatge “Press Space to begin” es va esborrant.

Warnings i possibles millores

1. Existeixen més modes de joc que no es poden arribar a usar ja que no es troben a les opcions del menú. Existeix codi relatiu a aquests modes de joc, però no està acabat d’implementar. Per exemple existeix un mode en el que només hi ha una pala horitzontal a l’estil Arkanoid [11].

2. La fase d’establiment de la connexió entre client i servidor és bastant millorable. El client envia un paquet al servidor, que està esperant. Aquest rep el paquet, i li contesta. Després d’aquesta fase comença el joc. A vegades si qualsevol dels dos falla en rebre el paquet per qualsevol motiu, ja no s’estableix la connexió i ambdós es queden esperant en bucles infinits. A més el jugador que fa de servidor haurà de començar la fase de connexió primer, perquè sinó també fallarà en rebre el paquet del client. Una solució seria que si passat un temps, després de l’enviament, no s’obté la confirmació de l’arribada es tornés a enviar el paquet.

3. El joc usa com a forma de control només el teclat encara que està implementada una opció per usar el ratolí però que no es pot escollir. Donat que la idea del joc és que sigui multi-plataforma seria bona idea afegir la opció de SDL_Joystick, ja que algunes de les plataformes amb les que hem treballat tenien la interfície SDL_Joystick portada (i cap de les plataformes tenia teclat). A més de que afegiríem una opció més de control també per al jugador de PC.

4. Els missatges que apareixen al finalitzar el joc i que indiquen al jugadors si han guanyat o perdut no s’entenen, ”Moshe says: YOU SUCK”, ”Moshe shuts the fuck up”. Seria més senzill usar els típics ”You win”, ”You lose”. A més els missatges són ofensius.

5. Per últim tenim els warnings que produeix la compilació, que podem consultar a la figura B.1 als apèndixs. Podem veure que la majoria d'ells són causats perquè es defineixen constants i variables que després no s'usen, o bé perquè s'usa una conversió d'`string` a `char*` que està obsoleta.

5.5.5 Compilació

Compilant la versió de Nintendo DS

Ara que ja tenim el codi del 2Pong modificat i preparat, podem compilar-lo. En aquesta secció veurem com preparar l'entorn de compilació per a la plataforma Nintendo DS. Aquest originalment està preparat només per compilar per a PC, per tant haurem d'afegir alguns canvis, com per exemple afegir nous `Makefile`, que es detallen a continuació.

A l'apartat 5.5.2 hem vist que hem hagut de generar un programa propi pel processador ARM7, per tant el procés de compilació serà una mica diferent de l'habitual: haurem de compilar per separat un programa per cada processador i després produir un únic fitxer `.nds` amb tota la informació. Per tant seran tres arxius `Makefile`: un per cada processador, i un per cridar als altres dos i ajuntar-los.

La manera més senzilla de fer-ho es recórrer a una plantilla que estigui dissenyada per aquest propòsit (compilar pels dos processadors). Tenim una entre els exemples del devkitPro. En concret la plantilla anomenada `combined` en el directori:

```
nds-examples-20100313/templates/combined
```

Nosaltres la utilitzarem de guia, però haurem de fer canvis per adaptar-la a les nostres necessitats.

Makefile principal

El fitxer `Makefile` del directori base és l'encarregat de cridar els altres dos `Makefile` (un per cada processador), i després crear el fitxer `.nds` amb els dos resultats. Per fer-ho utilitza l'eina `ndstool`:

```
ndstool -c $(TARGET).nds -7 $(TARGET).arm7 -9 $(TARGET).arm9
```

El seu codi complet el podem veure a la figura B.2 als apèndixs.

El podem copiar al directori base on tinguem el 2Pong, en el meu cas concret el directori `2Pong-Source`. Si ja tenim el `Makefile` de PC en aquest directori, li haurem de canviar de nom abans de copiar-lo per a que no coincideixin els noms. Li podem posar per exemple `Makefile.ds`. En cas que li canviem el nom, per compilar ho haurem de fer amb la comanda:

```
make -f Makefile.ds
```

En aquest arxiu podem canviar el nom del `TARGET` pel nom que li volem donar a l'arxiu final. Actualment està programat per a que agafi el nom del directori actual, si ho volem així no fa falta que ho canviem.

També podem copiar ara els altres dos `Makefile`. L'arxiu:

`nds-examples-20100313/templates/combined/arm7/Makefile`

el copiem a `2Pong-Source/arm7/` i l'arxiu:

`nds-examples-20100313/templates/combined/arm9/Makefile`

al directori `2Pong-Source/src/`.

Si els hem copiat en aquests directoris haurem de canviar al codi del `Makefile` principal la ubicació del `Makefile` d'ARM9, ja que la seva ubicació per omissió és `arm9/` i nosaltres l'hem col·locat a `src/`, que és on tenim el codi. Per tant haurem de canviar les línies:

```
$(TARGET).arm7   : arm7/$(TARGET).elf
$(TARGET).arm9   : src/$(TARGET).elf

src/$(TARGET).elf:
        $(MAKE) -C src

clean:
        $(MAKE) -C src clean
```

Per últim, podem fer que el joc estigui representat per l'icona del 2Pong afegint a la comanda ndstool:

```
ndstool ... -b $(CURDIR)/src/dades2Pong/iconDS.bmp "2PongDS;;"
```

Makefile ARM7

Al `Makefile` de l'ARM7, l'únic canvi que s'ha de fer és aquesta línia:

```
LIBS    := -ldswifi7 -lnds7
```

Per tal d'enllaçar les llibreries necessàries.

Makefile ARM9

Per últim, al `Makefile` de l'ARM9 s'han de modificar les següents línies:

```
SOURCES := .

CFLAGS  += $(INCLUDE) -DARM9 -I$(DEVKITPRO)/libnds/include/SDL

LIBS    := -lxml2 -lfat -lSDL_net -lSDL_mixer -lSDL -lm -ldswifi9 -lnds9
```

Per tal d'indicar on està el codi (directori actual), d'indicar el directori dels headers d'SDL i d'enllaçar les llibreries necessàries.

Compilant la versió de PSP

Igual que en la plataforma Nintendo DS, el toolchain ens prové d'una plantilla per compilar els nostres projectes. En concret utilitzarem una plantilla per compilar arxius elf, que són els executables de PSP. Aquesta plantilla la podem trobar als directoris del toolchain que ens vam descarregar, exactament a:

`toolchain/pspsdk/src/samples/template/elf_template`

Haurem de copiar el Makefile que hi ha dins aquest directori al directori del 2Pong, i fer algunes modificacions. El Makefile quedarà com el de la figura 5.3.

```
TARGET = 2PongPSP # Nom del programa generat
OBJS = src/ai.o src/balls.o ... # Tots els objectes que volem compilar

INCDIR =
CFLAGS = -O2 -G0 -Wall -I$(PSPDEV)/psp/include/libxml2 $(shell $(PSPBIN)/sdl-config --
    cflags) # Afegim directoris capc,aleres
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti
ASFLAGS = $(CFLAGS)

LIBDIR =
LDFLAGS =

LIBS += -lxml2 -lSDL_net -lm -lz -lstdc++ $(shell $(PSPBIN)/sdl-config --libs) # Afegim
    llibreries a enllac,ar

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = 2PongPSP # Titol que es mostrara al menu de PSP

PSPSDK=$(shell psp-config --pspsdk-path)
PSPBIN = $(PSPSDK)/../bin
include $(PSPSDK)/lib/build.mak
```

Figura 5.3: Makefile del 2Pong per PSP.

5.6 Depuració i proves

Nintendo DS

A l'hora de fer proves tenim dues opcions: provar al hardware final o en un emulador. Finalment no s'ha usat cap emulador perquè com ja vam veure a la fase d'anàlisi (secció 3.7) no es podia fer servir la connexió sense fil, necessària pel nostre programa. Per tant vam haver de provar al hardare final, que té el desavantatge que és molt costós en temps ja que per cada prova s'ha d'extreure la targeta MicroSD i connectar-la al PC.

Majoritàriament s'han fet proves amb una Nintendo DS Lite amb un flashcart M3 DS Real [66] però també s'han fet proves amb altres models de consoles i flashcart, incloent un Supercard (un model de Slot-2) [107].

Per fer depuració del codi també tenim diverses opcions: La primera, si utilitzem emulador, es pot enllaçar l'emulador DeSmuME amb GDB [41], el depurador estàndard de GNU. Es va provar aquesta opció, però com que finalment no s'ha usat emulador, no la descriurem a fons. Tenim un manual a [40] per si el lector està interessat.

L'altra opció per depurar és mostrar missatges per pantalla a la mateixa consola. Com que SDL només usa una de les dues pantalles de Nintendo DS, la de dalt, podem usar l'altra pantalla per mostrar missatges. El codi és de la llibreria libnds, per tant depenent de la plataforma.

```
consoleDebugInit(DebugDevice_CONSOLE);
fprintf(stderr, "debug message on DS console screen");
```

PSP

Quant a PSP tampoc s'ha usat emulador, s'han fet proves directament a una PSP-1000 amb un firmware CF 5.00 M33-4. No s'ha pogut provar en cap altre model per no disposar d'un exemplar alliberat. Per copiar el joc creat es connecta la consola al PC a través d'un cable USB, el qual també es un procés bastant lent.

Per fer depuració de codi, no podíem disposar de missatges per consola com a Nintendo DS. PSPSDK ens ofereix una utilitat de consola [21], però és per a la visualització amb la interfície GU i no SDL, a més que com que només disposem d'una pantalla, impediria la visualització correcta del joc.

Per tant vam haver d'utilitzar un fitxer per escriure missatges. Aquest també és un procés molt lent perquè cada vegada que vulguem consultar el resultat de l'execució haurem de tornar a connectar la consola al PC i obrir el fitxer.

Multi-plataforma

Finalment indicar que ens hem recolzat en altres eines a l'hora de fer proves. Per exemple, s'ha usat molt el Wireshark [117], un programa lliure per capturar paquets de la xarxa, a l'hora de controlar els paquets que s'enviaven els diferents dispositius, per exemple per controlar la fase de connexió entre client i servidor, o per generar els paquets DNS. Ha estat una eina clau per trobar errors.

5.7 Distribució del codi

Donat que aquest projecte s'ha intentat realitzar utilitzant només eines lliures també volíem que el codi generat ho fos. Per seguir amb la filosofia del programari lliure hem decidit distribuir tots els canvis i millores que hem fet en els codis, per a que aquests puguin ser usats i millorats si algú ho desitja.

Hem allotjat el codi a Google code ja que ens ofereix un entorn col·laboratiu còmode per projectes de codi obert o programari lliure. Aquest sistema ens ofereix un repositori amb 2 GB de capacitat per allotjar el codi, amb control de versions (subversion o mercurial), seguiment de qüestions relacionades amb el projecte (*issue tracker*), una wiki i una secció de descàrregues.

Hem anomenat el nostre projecte "2pong-multiplatform" i ja hi podem trobar tot el codi generat. Està disponible a [2].

Per descarregar el codi ho podem fer per exemple mitjançant el sistema de control de versions Subversion [9], amb la comanda:

```
svn checkout http://2pong-multiplatform.googlecode.com/svn/trunk/ 2pong-
    multiplatform
```

Encara que també es pot descarregar de la secció Downloads.

Capítol 6

Valoració econòmica

Un dels costos que comporta un projecte d'aquest tipus és l'adquisició del material necessari per a la realització del projecte. A la taula 6.1 podem consultar les despeses que suposaria adquirir aquests materials. S'ha de tenir en compte que els costos fan referència als preus en el seu moment d'adquisició, poden variar respecte als preus actuals.

El programari usat és completament lliure, per tant no comporta cap cost addicional.

Encara que sigui habitual en un projecte, no es va fer una planificació prèvia molt exhaustiva. Això és degut a que el nostre projecte implementava una idea de la que teníem suposicions que funcionaria però no en teníem proves. A més no havíem treballat prèviament amb aquests dispositius i no sabíem amb quins problemes ens trobaríem. Per últim volíem tenir una porta oberta per a possibles modificacions en els plans originals. No obstant ens hem esforçat en fer un recompte real d'hores.

Per a la realització d'aquest projecte s'hi han dedicat aproximadament unes 771 hores, que podem veure a la taula 6.2 com es distribueixen segons les tasques que s'han realitzat.

Concepte	**Descripció**	**Preu**
Nintendo DS Lite		150 €
Flashcart per a NDS	M3 DS Real + MicroSD 4 GB SDHC	35 €
PSP	Model PSP-1000, descatalogat. Incloïa Memory Stick i Cable USB	250 €
Ordinador portàtil	Samsung Q210, CPU Intel Core 2 Dup P8400 2.26 GHz, 3GB RAM DDR2 amb GNU/Linux Ubuntu 10.04 LTS Lucid Lynx	650 €
Router Wi-Fi	Linksys WRT120N	50 €
Total		**885 €**

Taula 6.1: Cost que suposaria el material utilitzat.

Concepte	**Hores**
Nintendo DS	307
Estudi de la plataforma i dels entorns de desenvolupament	23
Preparar un entorn devkitPro	27
Compilar i provar ports de SDL: GPF, GsoC, SDL_mixer, SDL_image	41
Portar SDL_net i funció gethostbyaddr	69
Portar libxml2	16
Adaptar codi 2Pong: Problemes resolució pantalla	32
Adaptar codi 2Pong: Adaptar controls i arreglar pantalla tàctil	35
Adaptar codi 2Pong: Altres	64
PSP	108
Estudi de la plataforma i dels entorns de desenvolupament	12
Alliberament del dispositiu	9
Instal·lar toolchain i llibreries necessàries	9
Adaptar codi 2Pong: Utilitzar diàlegs natius de PSP amb SDL	22
Adaptar codi 2Pong: Problemes de connexió	19
Adaptar codi 2Pong: Altres	37
2Pong i entorn de desenvolupament	100
Cercar i triar el joc	15
Estudiar el codi	17
Arreglar errors generals en el codi	23
Estudiar i instal·lar Scratchbox	25
Estudiar SDL	20
Documentació i altres	256
Instal·lar i aprendre LaTeX	12
Escriure informe previ	13
Escriure memòria	196
Reunir-nos amb el director	35
Total	**771**

Taula 6.2: Distribució d'hores treballades segons tasques realitzades.

Capítol 7

Conclusions i treball futur

7.1 Conclusions

L'objectiu inicial del nostre projecte era portar un videojoc lliure i multi-plataforma a Nintendo DS i iPhone i permetre que els dos dispositius interaccionessin en un mode de joc en xarxa. Podem concloure que aquest objectiu ha estat assolit, ja que disposem d'una versió jugable del joc en els dos dispositius i la comunicació entre ells funciona. Per tant podem dir que hem trencat les barreres del màrqueting de les que parlàvem a la introducció.

A més, durant la realització d'aquest projecte s'han assolit altres objectius que no estaven previstos al començament com ara portar llibreries, aplicar millores de codi o fins i tot portar el joc a una altra plataforma, PSP.

A nivell personal m'ha aportat molts coneixements. Gràcies a aquest projecte he pogut treballar amb plataformes que no havia pogut provar durant el transcurs dels estudis, i que tenia moltes ganes d'aprendre com era la programació per aquests dispositius. A més he tingut l'oportunitat d'endinsar-me dins el món del programari lliure, i que crec que és una bona filosofia molt beneficiosa per a tothom i que a partir d'ara seguiré utilitzant. També he descobert LaTeX, que és una excel·lent manera d'escriure documentació que s'adequa molt bé a aquest tipus de treballs tècnics. Per últim, descobrir i aprendre SDL ha estat una experiència molt enriquidora que estic segur que em servirà de molt en el futur. Estic agraït per totes les coses que m'ha descobert aquest treball.

Aquest projecte s'ha afavorit de la filosofia del programari lliure, i també es pretén contribuir a la comunitat. El port de SDL_net per a Nintendo DS és un exemple d'aquesta contribució, ja que a Internet no es pot trobar cap port. Nosaltres hem compilat la llibreria per a la plataforma, i així qualsevol persona el podrà utilitzar per fer programes amb xarxa.

Per acabar m'agradaria extreure algunes conclusions de la feina realitzada.

Primer alguns comentaris sobre el procés de recerca d'informació. En un projecte d'aquestes característiques la informació canvia ràpidament: els entorns de desenvolupament es

van actualitzant, surten noves eines, etc. És per això que quasi tota la informació es troba a la xarxa, molt poca informació relacionada amb aquest projecte es pot trobar als llibres. A més la informació sobre tecnologies aviat queda obsoleta, però en aquest món sembla que encara més.

Però aquest ritme frenètic dels entorns de desenvolupament fa que les llibreries i codi que utilitzem aviat es quedin obsolets i això suposa un problema. Per exemple, la llibreria libnds va canviant constantment el seu codi i si l'actualitzes a noves versions, és possible que hagin canviat funcions o estructures i el teu codi s'hagi de modificar per adaptar-lo, com hem vist vàries vegades en aquest document. A més, la informació reflectida en aquest document pot quedar antiquada perquè fa referència a totes aquestes eines que continuaran evolucionant.

No ha estat aquesta la única dificultat que ens han imposat les eines. També la incompatibilitat entre versions, per exemple muntar un entorn PAlib amb devkitPro va ser complicat. A més el port de SDL per Nintendo DS és bastant millorable i ha donat molts problemes. Sempre que he fet una consulta relacionada amb aquest port, m'han recomanat que no l'utilitzi.

La xarxa però es converteix en una eina indispensable de cerca d'informació, on a més es pot obtenir ajuda d'altres mitjançant fòrums, *mailing lists*, etc. En aquests desenvolupaments per a consoles portàtils, son molt útils les comunitats de desenvolupadors que es creen en torn a les consoles, com per exemple [97], una comunitat de PSP amb molta activitat a la que jo he recorregut per consultar durant la realització d'aquest projecte. Hem de tenir en compte que són plataformes tancades, i que per tant és difícil obtenir resultats, així si la gent no col·laborés no s'avançaria tant.

Per una altra banda, teníem dues opcions per portar un joc: el podíem haver implementat nosaltres des de zero, o podíem buscar un joc lliure. Aquesta última opció tenia l'avantatge que ens estalviava el temps d'implementació i ens permetia centrar-nos en altres coses; però l'inconvenient de que ens ha condicionat molt ja que el codi tenia algunes errades, com per exemple que el codi era molt depenent de la resolució, i això el feien menys portable.

Com que he treballat amb dos dispositius també puc concloure que hi ha plataformes en les que és més fàcil treballar que d'altres. Podem observar que per fer el mateix port la dedicació d'hores ha estat molt més gran a Nintendo DS que a PSP. Però la realitat és que cada plataforma té uns avantatges i uns inconvenients. Per exemple, la PSP s'ha de desbloquejar per poder executar programes propis, i la Nintendo DS no inclou aquesta dificultat. En canvi, en PSP va ser molt més fàcil instal·lar l'entorn de desenvolupament i s'ofereix molt més suport per SDL, que a més ve integrada amb el toolchain.

Programar per videoconsoles és divertit perquè és molt diferent al que estem acostumats: és molt proper al hardware, no hi ha sistema operatiu, etc. És una bona manera d'iniciar-se en el desenvolupament per sistemes encastats. Per no mencionar que veure el resultat del teu propi treball funcionant en una videoconsola és molt satisfactori.

7.2 Possibles millores i ampliacions

Com en tot projecte teníem un temps limitat i per tant no hem pogut fer totes les coses que ens hauria agradat. Per aquest motiu vam haver de donar prioritat a algunes tasques sobre d'altres i és per això que encara queda marge per a un futur treball a partir d'aquesta entrega. A més, el fet que tot el programari creat sigui lliure fa que qualsevol pugui continuar treballant-hi. A continuació s'enumeren una sèrie de possibles ampliacions.

- Millorar el port de SDL utilitzat per Nintendo DS, i actualitzar-lo a les últimes versions de la llibreria libnds. Ja hem fet una mica de feina en aquest sentit però el port encara es podria millorar força. Els següents són els aspectes que jo he experimentat i que es podrien millorar: un mal aprofitament de memòria, problemes de comunicació entre els dos processadors de la consola, no funcionen l'àudio, la pantalla tàctil ni la interfície SDL_Mouse, i la interfície SDL_Joystick es podria millorar. Com he dit, aquests són els problemes que he experimentat jo, però potser hi ha més. Fins i tot es podrien proposar ampliacions com per exemple suport per doble pantalla.

- Acabar de portar la resolució DNS per a la Nintendo DS. Aquesta feina l'hem descrit a la secció 5.2.2, i si fem memòria el problema és que resol correctament adreces públiques però pot fallar en resoldre adreces locals. Seria molt profitós continuar treballant en aquest aspecte, ja que és una funcionalitat que contribueix a la comunitat, ja que amb aquesta funció es completa encara més la llibreria dswifi, àmpliament utilitzada. I també ens serveix per a que SDL_net es pugui portar completament, essent SDL_net també una bona contribució.

- El codi del 2Pong també accepta moltes millores encara. Per exemple, la versió de Nintendo DS no té àudio, i a la PSP es podria millorar el rendiment. També es podria ampliar el joc afegint-hi noves funcionalitats no dependents de la plataforma, com per exemple:

  - Millores gràfiques.
  - Acabar d'implementar algunes funcionalitats que no estan acabades, com per exemple controlar el joc amb el ratolí, o un nou mode e joc en què es pot jugar en horitzontal a l'estil Arkanoid.
  - Incloure més opcions al menú com per exemple per desactivar els power-ups, la possibilitat d'afegir un port de connexió per omissió, etc.

  Són només alguns suggeriments, però es podria explotar encara molt la idea.

- Per acabar es podria portar el joc a més plataformes i de tipus més variats, per així experimentar amb nous entorns de treball. Algunes idees són: una plataforma més oberta com GP2X Caanoo [48], alguna plataforma amb Android [8] o potser a alguna videoconsola de sobretaula com Wii [114].

Apèndix A

Instal·lació i manual del joc

A.1 Instal·lació

Instal·lació per a Nintendo DS

La instal·lació de Nintendo DS dependrà del flashcart que tinguem.

Haurem de copiar l'executable "`2PongDS.nds`" al directori d'instal·lació d'executables de Nintendo DS del nostre flashcart. En el cas del "M3 DS Real" van al directori "`/NDS`".

A més, haurem de copiar les dades del joc per separat. Copiem el directori "`dades2Pong`" que està ubicat a "`2Pong-Source/src`" a l'arrel de la nostra targeta de Nintendo DS.

Podem assegurar-nos que hem instal·lat bé comprovant que tenim una estructura com la següent:

```
/NDS/2Pong.nds
/dades2Pong/...
```

Instal·lació per a PSP

Podem copiar tot el directori "`2Pong-Source`" a "`/PSP/GAME/`". Els jocs homebrew han d'estar en aquest directori. És el mètode més ràpid encara que estarem copiant coses innecessàries com per exemple el codi . No obstant, els arxius que realment ens interessen són:

```
/PSP/GAME/2PongPSP/2PongPSP.elf
/PSP/GAME/2PongPSP/EBOOT.PBP
/PSP/GAME/2PongPSP/PARAM.SFO
/PSP/GAME/2PongPSP/src/data/...
```

A.2 Manual del joc

El joc comença amb el menú principal, on podem triar entre els dos modes de joc principals: Regular (un jugador) i Network (multi-jugador en xarxa). Per saber com navegar pels menús, es pot seguir l'esquema 4.1 que hem vist durant l'anàlisi del joc.

Si entrem en el mode Network, tant el servidor com el client hauran d'introduir el port que volen usar, i el client a més haurà d'escriure l'adreça IP i el port del servidor separats per dos punts. El port ha d'estar entre 1024 i 65535. En comptes d'una adreça IP també podem fer servir un hostname, com per exemple localhost. Per exemple:

- El servidor introdueix el port **5555** (i té adreça 192.168.1.33).
- El client introdueix el port **6666**. Després introdueix les dades del servidor **192.168.1.33:5555**.

Cal remarcar que si volem fer una partida en xarxa el primer en començar la partida i introduir les dades haurà de ser el servidor.

El joc implementa un seguit de power-ups:

DoubleSpeed Fa que la velocitat de la bola es dupliqui.

HalfSpeed Fa que la velocitat de la bola es redueixi a la meitat.

IncSpeed Incrementa un 25% la velocitat de la bola.

DecSpeed Decrementa un 25% la velocitat de la bola.

ChangePos Mou la bola a una posició aleatòria de la pantalla.

Split Duplica la bola.

Cal dir que en el mode d'un jugador controlem la pala de la dreta, i en el mode en xarxa controlem la pala dreta si fem de servidor i l'esquerra si fem de client.

Controls de la versió per a Nintendo DS

Usarem els botons i el pad direccional per controlar tant els menús com el propi joc. El botó A s'utilitza per confirmar i el botó B per cancel·lar. Es pot pausar el joc amb el botó Start.

El teclat per pantalla es controla amb la pantalla tàctil, amb la peculiaritat que haurem de prémer la tecla de punt i coma (;) per escriure dos punts (:).

Controls de la versió per a PSP

Usarem els botons i el pad direccional per controlar tant els menús com el propi joc. El botó ◯ s'utilitza per confirmar i el botó X per cancel·lar. Es pot pausar el joc amb el botó Start.

Sortiran els diàlegs natius de PSP tant per connectar a la xarxa com per escriure per pantalla.

Apèndix B

Codi

```
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o defs.o defs.cpp
'-mcpu=' is deprecated. Use '-mtune=' or '-march=' instead.
defs.h: In constructor 'defines::defines()':
defs.h:242: warning: defines::powerups' will be initialized after
defs.h:240: warning:   'int defines::windowheight'
defs.cpp:168: warning:   when initialized here
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o CSpriteBase.o CSpriteBase.cpp
'-mcpu=' is deprecated. Use '-mtune=' or '-march=' instead.
CSpriteBase.cpp: In member function 'int CSpriteBase::init(char*)':
CSpriteBase.cpp:24: warning: ignoring return value of 'char* fgets(char*, int, FILE*)',
    declared with attribute warn_unused_result
CSpriteBase.cpp:33: warning: ignoring return value of 'char* fgets(char*, int, FILE*)',
    declared with attribute warn_unused_result
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o CSprite.o CSprite.cpp
'-mcpu=' is deprecated. Use '-mtune=' or '-march=' instead.
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o font.o font.cpp
'-mcpu=' is deprecated. Use '-mtune=' or '-march=' instead.
font.cpp: In function 'SDLFont* initFont(char*, float, float, float, float)':
font.cpp:57: warning: ignoring return value of 'int fscanf(FILE*, const char*, ...)',
    declared with attribute warn_unused_result
font.cpp: In function 'void drawString(SDL_Surface*, SDLFont*, int, int, char*, ...)':
font.cpp:154: warning: array subscript has type 'char'
font.cpp:183: warning: array subscript has type 'char'
font.cpp:194: warning: array subscript has type 'char'
font.cpp: In function 'int stringWidth(SDLFont*, char*, ...)':
font.cpp:214: warning: array subscript has type 'char'
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o ai.o ai.cpp
'-mcpu=' is deprecated. Use '-mtune=' or '-march=' instead.
ai.cpp:11: warning: unused parameter 'difficulty'
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o physics.o physics.cpp
'-mcpu=' is deprecated. Use '-mtune=' or '-march=' instead.
```

```
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o net.o net.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
net.cpp: In function 'void UpdateClient(Game*, Ball*, Paddle*, _UDPsocket*, int)':
net.cpp:39: warning: unused variable 'num'
net.cpp: In function 'IPaddress GetAddress(SDL_Surface*, SDLFont*)':
net.cpp:165: warning: deprecated conversion from string constant to 'char*'
net.cpp:195: warning: deprecated conversion from string constant to 'char*'
net.cpp: In function 'int GetPort(SDL_Surface*, SDLFont*, int)':
net.cpp:228: warning: deprecated conversion from string constant to 'char*'
net.cpp:229: warning: deprecated conversion from string constant to 'char*'
net.cpp:243: warning: deprecated conversion from string constant to 'char*'
net.cpp:255: warning: deprecated conversion from string constant to 'char*'
net.cpp:256: warning: deprecated conversion from string constant to 'char*'
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o menu.o menu.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
menu.cpp: In function 'char* GetNodeValue(xmlNode*, char*)':
menu.cpp:13: warning: deprecated conversion from string constant to 'char*'
menu.cpp: In function 'xmlNode* GetPlayNode(xmlNode*, char*, char*)':
menu.cpp:23: warning: deprecated conversion from string constant to 'char*'
menu.cpp:24: warning: deprecated conversion from string constant to 'char*'
menu.cpp:25: warning: deprecated conversion from string constant to 'char*'
menu.cpp:26: warning: deprecated conversion from string constant to 'char*'
menu.cpp: In function 'void DrawMenu(xmlNode**, int, int, SDL_Surface*, SDLFont*)':
menu.cpp:65: warning: deprecated conversion from string constant to 'char*'
menu.cpp:67: warning: deprecated conversion from string constant to 'char*'
menu.cpp:69: warning: deprecated conversion from string constant to 'char*'
menu.cpp:71: warning: deprecated conversion from string constant to 'char*'
menu.cpp:79: warning: deprecated conversion from string constant to 'char*'
menu.cpp:80: warning: deprecated conversion from string constant to 'char*'
menu.cpp: In function 'xmlNode* ShowNode(xmlNode*, char*, SDL_Surface*, SDLFont*)':
menu.cpp:95: warning: deprecated conversion from string constant to 'char*'
menu.cpp:100: warning: deprecated conversion from string constant to 'char*'
menu.cpp: In function 'options* ParseNode(xmlNode*)':
menu.cpp:155: warning: deprecated conversion from string constant to 'char*'
menu.cpp:156: warning: deprecated conversion from string constant to 'char*'
menu.cpp:157: warning: deprecated conversion from string constant to 'char*'
menu.cpp:158: warning: deprecated conversion from string constant to 'char*'
menu.cpp:159: warning: deprecated conversion from string constant to 'char*'
menu.cpp: In function 'int InitMenu(char*, Game*, Ball*, Paddle*, Powerup*, defines,
    CSpriteBase*, Mix_Chunk*, Mix_Chunk*)':
menu.cpp:193: warning: deprecated conversion from string constant to 'char*'
menu.cpp:224: warning: deprecated conversion from string constant to 'char*'
menu.cpp:249: warning: deprecated conversion from string constant to 'char*'
menu.cpp:299: warning: deprecated conversion from string constant to 'char*'
menu.cpp:303: warning: deprecated conversion from string constant to 'char*'
menu.cpp:311: warning: deprecated conversion from string constant to 'char*'
menu.cpp:311: warning: deprecated conversion from string constant to 'char*'
menu.cpp:313: warning: deprecated conversion from string constant to 'char*'
menu.cpp:313: warning: deprecated conversion from string constant to 'char*'
menu.cpp:320: warning: deprecated conversion from string constant to 'char*'
menu.cpp:179: warning: unused variable 'aout'
menu.cpp: In function 'xmlNode* GetNode(xmlNode*, char*)':
menu.cpp:53: warning: control reaches end of non-void function
menu.cpp: In function 'int InitMenu(char*, Game*, Ball*, Paddle*, Powerup*, defines,
    CSpriteBase*, Mix_Chunk*, Mix_Chunk*)':
menu.cpp:184: warning: 'name' may be used uninitialized in this function
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o draw.o draw.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
draw.cpp: In function 'void DrawScene(Game*, Ball*, Paddle*, Powerup*, defines)':
draw.cpp:7: warning: unused variable 'font'
draw.cpp: In function 'void DrawResults(Game*, Paddle*, int, int)':
draw.cpp:49: warning: deprecated conversion from string constant to 'char*'
```

```
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o powerups.o powerups.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
powerups.cpp:8: warning: unused parameter 'mode'
powerups.cpp:23: warning: unused parameter 'mode'
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o balls.o balls.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
balls.cpp:23: warning: unused parameter 'difficulty'
balls.cpp:113: warning: unused parameter 'puk'
balls.cpp:113: warning: unused parameter 'pluck'
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o paddles.o paddles.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
paddles.cpp:51: warning: unused parameter 'balls'
paddles.cpp:51: warning: unused parameter 'difficulty'
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o game.o game.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
game.cpp:3: warning: unused parameter 'address'
game.cpp: In function 'void InitGame(Game*, Ball*, Paddle*, Powerup*, int, int, defines)':
game.cpp:178: warning: deprecated conversion from string constant to 'char*'
game.cpp:178: warning: deprecated conversion from string constant to 'char*'
game.cpp: In function 'void ResetGame(Game*, Ball*, Paddle*, Powerup*, int, int, defines)
    ':
game.cpp:201: warning: deprecated conversion from string constant to 'char*'
game.cpp:201: warning: deprecated conversion from string constant to 'char*'
game.cpp: In function 'int CheckState(Game*, Ball*, Paddle*, defines)':
game.cpp:223: warning: deprecated conversion from string constant to 'char*'
game.cpp:225: warning: deprecated conversion from string constant to 'char*'
game.cpp:228: warning: deprecated conversion from string constant to 'char*'
game.cpp:230: warning: deprecated conversion from string constant to 'char*'
game.cpp:256: warning: control reaches end of non-void function
game.cpp: In function 'int GameMain(Game*, int, Ball*, Paddle*, Powerup*, defines,
    CSpriteBase*, Mix_Chunk*, Mix_Chunk*, IPaddress, _UDPsocket*, int, int)':
game.cpp:16: warning: 'in' may be used uninitialized in this function
game.cpp:16: warning: 'out' may be used uninitialized in this function
g++ -c -pipe -fno-exceptions -Wall -W -O2 -march=i486 -mcpu=i686 -D_REENTRANT `xml2-config
     --cflags` `sdl-config --cflags` -o main.o main.cpp
`-mcpu=' is deprecated. Use `-mtune=' or '-march=' instead.
main.cpp:25: warning: unused parameter 'powerups'
main.cpp: In function 'void FirstInitPowerups(Powerup*, defines, CSpriteBase*)':
main.cpp:40: warning: deprecated conversion from string constant to 'char*'
main.cpp:41: warning: deprecated conversion from string constant to 'char*'
main.cpp:42: warning: deprecated conversion from string constant to 'char*'
main.cpp:43: warning: deprecated conversion from string constant to 'char*'
main.cpp:44: warning: deprecated conversion from string constant to 'char*'
main.cpp:45: warning: deprecated conversion from string constant to 'char*'
main.cpp:46: warning: deprecated conversion from string constant to 'char*'
main.cpp: At global scope:
main.cpp:36: warning: unused parameter 'powerups'
main.cpp:36: warning: unused parameter 'def'
main.cpp: In function 'void FirstInitBalls(Ball*, SDL_Surface*, defines)':
main.cpp:50: warning: deprecated conversion from string constant to 'char*'
main.cpp: In function 'void FirstInitPaddles(Paddle*, SDL_Surface*, defines)':
main.cpp:58: warning: deprecated conversion from string constant to 'char*'
main.cpp: In function 'int main(int, char**)':
main.cpp:117: warning: deprecated conversion from string constant to 'char*'
main.cpp:122: warning: deprecated conversion from string constant to 'char*'
main.cpp: At global scope:
main.cpp:76: warning: unused parameter 'argc'
main.cpp:76: warning: unused parameter 'argv'
g++ -fno-exceptions  -Wl,-rpath,/usr/lib/qt/lib -o 2pong defs.o CSpriteBase.o CSprite.o
    font.o ai.o physics.o net.o menu.o draw.o powerups.o balls.o paddles.o game.o main.o
    `xml2-config --libs` `sdl-config --libs` -lSDL_net -lSDL_mixer
mv 2pong ../
```

Figura B.1: Warnings del 2Pong

```
#---------------------------------------------------------------------------
.SUFFIXES:
#---------------------------------------------------------------------------
ifeq ($(strip $(DEVKITARM)),)
$(error "Please set DEVKITARM in your environment. export DEVKITARM=<path to>devkitARM")
endif

include $(DEVKITARM)/ds_rules

export TARGET           :=      $(shell basename $(CURDIR))
export TOPDIR           :=      $(CURDIR)


.PHONY: $(TARGET).arm7 $(TARGET).arm9

#---------------------------------------------------------------------------
# main targets
#---------------------------------------------------------------------------
all: $(TARGET).nds

#---------------------------------------------------------------------------
$(TARGET).nds   :       $(TARGET).arm7 $(TARGET).arm9
        ndstool -c $(TARGET).nds -7 $(TARGET).arm7 -9 $(TARGET).arm9

#---------------------------------------------------------------------------
$(TARGET).arm7  : arm7/$(TARGET).elf
$(TARGET).arm9  : arm9/$(TARGET).elf

#---------------------------------------------------------------------------
arm7/$(TARGET).elf:
        $(MAKE) -C arm7

#---------------------------------------------------------------------------
arm9/$(TARGET).elf:
        $(MAKE) -C arm9

#---------------------------------------------------------------------------
clean:
        $(MAKE) -C arm9 clean
        $(MAKE) -C arm7 clean
        rm -f $(TARGET).nds $(TARGET).arm7 $(TARGET).arm9
```

Figura B.2: Plantilla Makefile per compilar un programa arm7 i un arm9 i combinar-los.

```
#include <nds.h>
#include <dswifi7.h>

//-------------------------------------------------------------------------------
void VblankHandler(void) {
//-------------------------------------------------------------------------------
        Wifi_Update();
}

int vcount;
touchPosition first,tempPos;

//-------------------------------------------------------------------------------
void VcountHandler() {
//-------------------------------------------------------------------------------
        static int lastbut = -1;
        TransferRegion volatile * IPCmio = (TransferRegion volatile *)(0x027FF000);
        uint16 but=0, x=0, y=0, xpx=0, ypx=0, z1=0, z2=0;
        but = REG_KEYXY;
        if (!( (but ^ lastbut) & (1<<6))) {
                touchReadXY(&tempPos);
                if ( tempPos.rawx == 0 || tempPos.rawy == 0 ) {
                        but |= (1 <<6);
                        lastbut = but;
                } else {
                        x = tempPos.rawx;
                        y = tempPos.rawy;
                        xpx = tempPos.px;
                        ypx = tempPos.py;
                        z1 = tempPos.z1;
                        z2 = tempPos.z2;
                }
        } else {
                lastbut = but;
                but |= (1 <<6);
        }
        if ( vcount == 80 ) {
                first = tempPos;
        } else {
                if (abs( xpx - first.px) > 10 || abs( ypx - first.py) > 10 ||
                                (but & ( 1<<6)) ) {
                        but |= (1 <<6);
                        lastbut = but;
                } else {
                        IPCmio->touchX          = x;
                        IPCmio->touchY          = y;
                        IPCmio->touchXpx        = xpx;
                        IPCmio->touchYpx        = ypx;
                        IPCmio->touchZ1         = z1;
                        IPCmio->touchZ2         = z2;
                }
        }
        IPCmio->buttons         = but;
        vcount ^= (80 ^ 130);
        SetYtrigger(vcount);
}

volatile bool exitflag = false;

//-------------------------------------------------------------------------------
void powerButtonHandler() {
//-------------------------------------------------------------------------------
        exitflag = true;
}
```

```
//---------------------------------------------------------------------------
int main() {
//---------------------------------------------------------------------------
        rtcReset();
        readUserSettings();
        irqInit();
        // Start the RTC tracking IRQ
        initClockIRQ();
        fifoInit();
        SetYtrigger(80);
        vcount = 80;
        installWifiFIFO();
        installSoundFIFO();
        installSystemFIFO();
        irqSet(IRQ_VCOUNT, VcountHandler);
        irqSet(IRQ_VBLANK, VblankHandler);
        REG_IPC_FIFO_CR = IPC_FIFO_ENABLE | IPC_FIFO_SEND_CLEAR | IPC_FIFO_RECV_IRQ;
        irqEnable( IRQ_VBLANK | IRQ_VCOUNT | IRQ_NETWORK|IRQ_FIFO_NOT_EMPTY);
        // Keep the ARM7 mostly idle
        while (!exitflag) {
                swiWaitForVBlank();
        }
        return 0;
}
```

Figura B.3: Programa personalitzat per al processador ARM7 de la Nintendo DS.

```
user@laptop:~/sbox2/rootfs/nds/hello_world/source$ sb2
[SB2 simple nintendo-ds] user@laptop source $ ls
main.cpp
[SB2 simple nintendo-ds] user@laptop source $ gcc -MMD -MP -MF main.d -g -Wall -O2
-march=armv5te -mtune=arm946e-s -fomit-frame-pointer -ffast-math -mthumb -mthumb-interwork
-I$HOME/x-tools/devkitPro/libnds/include -DARM9 -fno-rtti -fno-exceptions -c main.cpp
-o main.o
[SB2 simple nintendo-ds] user@laptop source $ gcc -specs=ds_arm9.specs -g -mthumb
-mthumb-interwork -mno-fpu main.o -L$HOME/x-tools/devkitPro/libnds/lib -lnds9 -o main.elf
[SB2 simple nintendo-ds] user@laptop source $ objcopy -O binary main.elf main.arm9
[SB2 simple nintendo-ds] user@laptop source $ ndstool -c main.nds -9 main.arm9
Nintendo DS rom tool 1.41 - May  1 2009
by Rafael Vuijk, Dave Murphy, Alexei Karpenko
[SB2 simple nintendo-ds] user@laptop source $ ls
main.arm9  main.cpp  main.d  main.elf  main.nds  main.o
[SB2 simple nintendo-ds] user@laptop source $ file main.elf
main.elf: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked, not stripped
[SB2 simple nintendo-ds] user@laptop source $ exit
user@laptop:~/sbox2/rootfs/nds/hello_world/source$ desmume main.nds

Rom serial: Homebrew
```

Figura B.4: Exemple de compilació d'un programa per a Nintendo DS amb Scratchbox2

Bibliografia

[1] 2Pong. http://twopong.sourceforge.net/. [Online; Visitada per darrera vegada el 24 de novembre de 2010].

[2] 2pong-multiplatform. http://code.google.com/p/2pong-multiplatform/. [Online; Visitada per darrera vegada el 13 de gener de 2011].

[3] A FAT library for the Nintendo GBA and DS. http://chishm.drunkencoders.com/libfat/. [Online; Visitada per darrera vegada el 12 de novembre de 2010].

[4] A guide to homebrew development for the Nintendo DS. http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html. [Online; Visitada per darrera vegada el 9 de novembre de 2010].

[5] Advanced Packaging Tool. http://wiki.debian.org/Apt. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[6] All PSP Error Codes. http://www.psp-programming.com/forums/index.php/topic,538.0.html. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[7] Allegro - A game programming library. http://alleg.sourceforge.net/. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[8] Android.com. http://www.android.com/. [Online; Visitada per darrera vegada el 13 de gener de 2011].

[9] Apache Subversion. http://subversion.apache.org/. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[10] App Store metrics. http://148apps.biz/app-store-metrics/. [Online; Visitada per darrera vegada el 2 de gener de 2011].

[11] Arkanoid - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/Arkanoid. [Online; Visitada per darrera vegada el 8 de gener de 2011].

[12] arpa/inet.h Specification. http://opengroup.org/onlinepubs/007908775/xns/arpainet.h.html. [Online; Visitada per darrera vegada l'1 de desembre de 2010].

[13] Autres Librairies - libxml2. http://www.smartpsp.net/programmation-sur-psp-librairies/autres-librairies/libxml2.html. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[14] Chroot - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/w/index.php?title=Chroot&oldid=42505152. [Online; Visitada per darrera vegada el 8 de gener de 2011].

[15] Common syntax errors in C++. http://womble.decadent.org.uk/c++/syntax-errors.html. [Online; Visitada per darrera vegada el 7 de gener de 2011].

[16] Creative Commons. Reconocimiento No Comercial 3.0 España. http://creativecommons.org/licenses/by-nc/3.0/es/. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[17] Cross Compiler - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Cross_compiler. [Online; Visitada per darrera vegada el 22 de desembre de 2010].

[18] Cross-platform — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Cross-platform&oldid=405027135. [Online; Visitada per darrera vegada el 2 de gener de 2011].

[19] Custom Firmware M33. http://psp.scenebeta.com/noticia/custom-firmware-m33. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[20] Danzeff OSK Source Released. http://www.qj.net/psp/homebrew-development/danzeff-osk-source-released.html. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[21] Debug Utility Library. http://psp.jim.sh/pspsdk-doc/group__Debug.html. [Online; Visitada per darrera vegada el 16 de gener de 2011].

[22] DeSmuME Nintendo DS Emulator. http://desmume.org/. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[23] devkitPro. http://www.devkitpro.org/. [Online; Visitada per darrera vegada el 10 de novembre de 2010].

[24] devkitPro Project at SourceForge. http://sourceforge.net/projects/devkitpro/. [Online; Visitada per darrera vegada el 25 de novembre de 2010].

[25] devkitPSP at SourceForge. http://sourceforge.net/projects/devkitpro/files/devkitPSP/. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[26] DLDI Wiki Main Page. http://dldi.drunkencoders.com/. [Online; Visitada per darrera vegada el 15 de novembre de 2010].

[27] DS Cards Legality. http://www.nds-gear.com/ds-dsi-cards-legal-legality. [Online; Visitada per darrera vegada el 2 de novembre de 2010].

[28] DS Technical Data. http://nocash.emubase.de/gbatek.htm#dstechnicaldata. [Online; Visitada per darrera vegada el 23 de novembre de 2010].

[29] DSEmu Nintendo DS Emulator. http://www.ndsemulator.com/nintendo-ds/dsemu.htm. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[30] DSi and WPA. http://devkitpro.org/viewtopic.php?f=23&t=1561. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[31] dswifi: Wifi for DS Homebrew Development. http://www.akkit.org/dswifi/index.html. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[32] Dualis Nintendo DS Emulator. http://dualis.1emu.net/. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[33] EFS Library. http://nodadev.wordpress.com/nds-projects/efs-library/. [Online; Visitada per darrera vegada el 15 de novembre de 2010].

[34] Ensata Nintendo DS Emulator. http://www.intsys.co.jp/english/. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[35] FlashCarts de NDS. http://www.elotrolado.net/wiki/Flashcarts_de_NDS. [Online; Visitada per darrera vegada el 3 de novembre de 2010].

[36] Free software. http://www.gnu.org/philosophy/free-sw.html. [Online; Visitada per darrera vegada el 2 de gener de 2011].

[37] Galaxy Gameworks. http://www.galaxygameworks.com/index.html. [Online; Visitada per darrera vegada el 12 de gener de 2011].

[38] Game Boy Advance - Viquipèdia, l'Enciclopèdia Lliure. http://ca.wikipedia.org/wiki/Game_Boy_Advance. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[39] GCC, the GNU Compiler Collection. http://gcc.gnu.org/. [Online; Visitada per darrera vegada el 9 de gener de 2011].

[40] GDB and Desmume. http://code.google.com/p/quirkysoft/wiki/DebuggingNotes. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[41] GDB: The GNU Project Debugger. http://www.gnu.org/software/gdb/. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[42] gethostbyaddr function. http://beej.us/guide/bgnet/output/html/multipage/gethostbynameman.html. [Online; Visitada per darrera vegada l'1 de gener de 2011].

[43] GNU Binutils. http://www.gnu.org/software/binutils/. [Online; Visitada per darrera vegada el 9 de gener de 2011].

[44] GNU C Library. http://www.gnu.org/software/libc/. [Online; Visitada per darrera vegada el 9 de gener de 2011].

[45] GNU General Public License. http://www.gnu.org/licenses/gpl.html. [Online; Visitada per darrera vegada el 2 de gener de 2011].

[46] GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html. [Online; Visitada per darrera vegada el 2 de gener de 2011].

[47] Google Summer of Code 2008. http://code.google.com/p/google-summer-of-code-2008-sdl/. [Online; Visitada per darrera vegada el 20 d'octubre de 2010].

[48] GP2X Caanoo - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/GP2X_Caanoo. [Online; Visitada per darrera vegada el 13 de gener de 2011].

[49] GPF's Development Site. http://gpf.dcemu.co.uk/ndsSDL.shtml. [Online; Visitada per darrera vegada el 20 d'octubre de 2010].

[50] Homebrew - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/Homebrew. [Online; Visitada per darrera vegada el 25 d'octubre de 2010].

[51] Homebrew - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Nintendo_DS_homebrew. [Online; Visitada per darrera vegada el 2 de novembre de 2010].

[52] Homebrew en PlayStation Portable - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/Homebrew_en_PlayStation_Portable. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[53] Homebrew Menu. Cards that support the ARGV protocol. http://devkitpro.org/wiki/Homebrew_Menu#Cards_that_support_the_ARGV_protocol. [Online; Visitada per darrera vegada el 15 de novembre de 2010].

[54] How to run code. http://tobw.net/dswiki/index.php?title=How_to_run_code. [Online; Visitada per darrera vegada el 2 de novembre de 2010].

[55] How To Setup Cross Compile Environment. http://biffengineering.com/wiki/index.php?title=HowToSetupCrossCompileEnvironment. [Online; Visitada per darrera vegada el 8 de gener de 2011].

[56] iDeaS Nintendo DS Emulator. http://ideasemu.biz/index.php. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[57] IEEE 802.11. http://standards.ieee.org/getieee802/download/802.11-2007.pdf. [Online; Visitada per darrera vegada el 2 de gener de 2011].

[58] Introduction to Nintendo DS Programming. http://patater.com/files/projects/manual/manual.html. [Online; Visitada per darrera vegada l'11 de novembre de 2010].

[59] keyboard.h File Reference. http://libnds.devkitpro.org/a00125.html. [Online; Visitada per darrera vegada el 12 de gener de 2011].

[60] LaTeX – A document preparation system. http://www.latex-project.org/. [Online; Visitada per darrera vegada el 10 de gener de 2011].

[61] Leyendo la entrada de PSP. http://es.wikibooks.org/wiki/Desarrollo_de_videojuegos_para_PSP_con_C%2B%2B_y_SDL/Leyendo_la_entrada_de_PSP. [Online; Visitada per darrera vegada el 16 de gener de 2011].

[62] Libnds Documentation. http://libnds.devkitpro.org/. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[63] libxml2 - The XML C parser and toolkit of Gnome. http://xmlsoft.org/. [Online; Visitada per darrera vegada l'1 de desembre de 2010].

[64] Libxml2 - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Libxml2&oldid=397898647. [Online; Visitada per darrera vegada el 4 de gener de 2011].

[65] Licencia BSD - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/Licencia_BSD. [Online; Visitada per darrera vegada el 16 de gener de 2011].

[66] M3 DS Real. http://www.m3adapter.com/SLOT-1_Series_M3DS-R.htm. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[67] Maemo. http://maemo.org/. [Online; Visitada per darrera vegada el 6 de gener de 2011].

[68] Massively multiplayer online role-playing game - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Massively_multiplayer_online_role-playing_game. [Online; Visitada per darrera vegada el 4 de gener de 2011].

[69] Microprogramari - Viquipèdia, l'Enciclopèdia Lliure. http://ca.wikipedia.org/wiki/Microprogramari. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[70] NeonDS Nintendo DS Emulator. http://www.neonds.com/en. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[71] Newlib C library. http://sourceware.org/newlib/. [Online; Visitada per darrera vegada el 3 de gener de 2011].

[72] NightFox's Lib. Eliminado el soporte para EFS. Añadido soporte para NitroFS. http://blogs.gamefilia.com/knightfox/12-03-2010/31393/nightfoxs-lib-eliminado-el-soporte-para-efs-anadido-soporte-para-nitrofs. [Online; Visitada per darrera vegada el 15 de novembre de 2010].

[73] NightFox's Lib. http://www.nightfoxandco.com/index.php/programacion/nfl-es/. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[74] Nintendo 3DS - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Nintendo_3DS#3DS_vs_DS_Series. [Online; Visitada per darrera vegada el 21 d'octubre de 2010].

[75] Nintendo DS - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/w/index.php?title=Nintendo_DS&oldid=40950567. [Online; Visitada per darrera vegada el 20 d'octubre de 2010].

[76] Nintendo DS - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Nintendo_DS&oldid=391560419. [Online; Visitada per darrera vegada el 20 d'octubre de 2010].

[77] Nintendo DS Emulation - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Nintendo_DS_emulation. [Online; Visitada per darrera vegada el 18 de novembre de 2010].

[78] Nintendo DS Emulator. http://www.ndsemulator.com/. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[79] Nintendo DS Memory Layout. http://www.dev-scene.com/NDS/Tutorials_Day_2#Memory_Layout. [Online; Visitada per darrera vegada el 23 de novembre de 2010].

[80] Nintendo DS R4 cartridges ruled illegal. http://www.telegraph.co.uk/technology/nintendo/7916159/Nintendo-DS-R4-cartridges-ruled-illegal.html. [Online; Visitada per darrera vegada el 2 de novembre de 2010].

[81] Nintendo Software Development Support Group (SDK oficial de Nintendo). http://www.warioworld.com/. [Online; Visitada per darrera vegada el 30 d'octubre de 2010].

[82] Nitrofs Filesystem Driver for Nintendo DS. http://blea.ch/wiki/index.php/Nitrofs. [Online; Visitada per darrera vegada el 15 de novembre de 2010].

[83] no$gba Nintendo DS Emulator. http://nocash.emubase.de/gba.htm. [Online; Visitada per darrera vegada el 18 de novembre de 2010].

[84] No$GBA offers DS emulator . http://www.eurogamer.net/articles/news150806dsemu. [Online; Visitada per darrera vegada el 18 de novembre de 2010].

[85] Nokia. http://www.nokia.com/. [Online; Visitada per darrera vegada el 8 de gener de 2011].

[86] OSK and SDL. http://www.psp-programming.com/forums/index.php/topic,4983.0.html. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[87] PAlib 090621. http://www.dcemu.co.uk/vbulletin/showthread.php?t=218629. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[88] PAlib Community Development. http://www.palib-dev.com/. [Online; Visitada per darrera vegada el 12 de novembre de 2010].

[89] PAlib installation. http://palib-dev.com/wiki/doku.php?id=installation. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[90] PAlib Projects. http://palib-dev.com/wiki/doku.php?id=projects. [Online; Visitada per darrera vegada el 16 de novembre de 2010].

[91] PlayStation Portable - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/PlayStation_Portable. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[92] PlayStation Portable Technical Specifications - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/PlayStation_Portable#Technical_specifications. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[93] PlayStation Store - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/PlayStation_Store. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[94] Pong - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/Pong. [Online; Visitada per darrera vegada el 24 de novembre de 2010].

[95] Port numbers. http://www.iana.org/assignments/port-numbers. [Online; Visitada per darrera vegada el 7 de gener de 2011].

[96] PSP Programming. http://en.wikibooks.org/wiki/PSP_Programming. [Online; Visitada per darrera vegada el 16 de gener de 2011].

[97] PSP-Programming.com Forums. http://www.psp-programming.com/forums/. [Online; Visitada per darrera vegada el 13 de gener de 2011].

[98] PSPDEV Toolchain. http://wiki.ps2dev.org/psp:toolchain. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[99] PSPSDK. http://wiki.ps2dev.org/psp:pspsdk. [Online; Visitada per darrera vegada el 16 de gener de 2011].

[100] Quemu. http://wiki.qemu.org/Main_Page. [Online; Visitada per darrera vegada el 8 de gener de 2011].

[101] RFC 1035 - Domain names - implementation and specification. http://www.faqs.org/rfcs/rfc1035.html. [Online; Visitada per darrera vegada el 3 de gener de 2011].

[102] Scratchbox. http://www.scratchbox.org/. [Online; Visitada per darrera vegada el 6 de gener de 2011].

[103] SDL net Documentation. http://www.libsdl.org/projects/SDL_net/docs/index.html. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[104] SDL_net. http://www.libsdl.org/projects/SDL_net/. [Online; Visitada per darrera vegada el 27 de novembre de 2010].

[105] Simple DirectMedia Layer. http://www.libsdl.org/. [Online; Visitada per darrera vegada el 12 de novembre de 2010].

[106] Simple DirectMedia Layer - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Simple_DirectMedia_Layer. [Online; Visitada per darrera vegada el 22 de novembre de 2010].

[107] SuperCard Lite product introduction. http://eng.supercard.sc/manual/lite_jj.htm. [Online; Visitada per darrera vegada el 15 de gener de 2011].

[108] TeX - Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/TeX. [Online; Visitada per darrera vegada el 10 de gener de 2011].

[109] Toolchain - Wikipedia, La enciclopedia libre. http://es.wikipedia.org/wiki/Cadena_de_herramientas. [Online; Visitada per darrera vegada el 9 de novembre de 2010].

[110] Tutorial para portar juegos de PC a PSP en C++ y SDL. http://grannysbloodbath.wordpress.com/2010/06/18/tutorial-para-portar-juegos-de-pc-a-psp-en-c-y-sdl/. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[111] uClibc C library. http://www.uclibc.org/. [Online; Visitada per darrera vegada el 3 de gener de 2011].

[112] VGChartz Hardware Totals. http://www.vgchartz.com/hardware_totals.php. [Online; Visitada per darrera vegada el 21 d'octubre de 2010].

[113] WifiMe Tutorial. http://www.aaronrogers.com/nintendods/wifime.php. [Online; Visitada per darrera vegada el 3 de novembre de 2010].

[114] Wii - Nintendo. http://www.nintendo.es/NOE/es_ES/wii_54.html. [Online; Visitada per darrera vegada el 13 de gener de 2011].

[115] Wikibooks - Manual de LaTeX. http://es.wikibooks.org/wiki/LaTeX. [Online; Visitada per darrera vegada el 10 de gener de 2011].

[116] WineHQ. http://www.winehq.org/. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[117] Wireshark. http://www.wireshark.org/. [Online; Visitada per darrera vegada el 14 de gener de 2011].

[118] XML. http://www.w3.org/XML/. [Online; Visitada per darrera vegada el 4 de gener de 2011].

[119] Anna Monros Bellart. Videojocs lliures: Interacció multiplataforma, 2011. Facultat d'Informàtica de Barcelona (UPC).

[120] Gabriel Valiente Feruglio. *Composició de textos científics amb LATEX*. Edicions UPC, 1998.