EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi: Eng. Tècn. Informàtica de Gestió. Pla 2001

Títol: Desenvolupament d'un motor 2D per videojocs de plataformes mòbils

Document: Memòria

Alumne: Andrés Quesada Molinero

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

Convocatòria (mes/any): Setembre/2013

Índex de Continguts

1. Introducció

El sector dels jocs per a plataformes mòbils s'ha disparat des de el llançament de Iphone i les plataformes amb Android. Jocs com Angry Birds han aconseguit més de 200 milions de descàrregues i estan disponibles en quasi tots els sistemes operatius mòbils.

El mercat de les aplicacions mòbils en general està creixent a un ritme vertiginós. Segons un estudi de l'IDC, al 2015 hi haurà 182.700 milions de descàrregues d'apps a l'any. I els jocs si continua aquesta tendència, seran un dels tipus d'aplicació mes descarregada.

Segons dades del Chomp, un buscador d'aplicacions, el tipus d'aplicació més popular per a Android son els jocs, amb un 27,1% del mercat. En segon lloc es troben les apps d'utilitats i entreteniment amb un 16% cadascuna.

A més, els usuaris estan disposats a pagar per aquest tipus d'aplicacions. Un estudi de *Nielsen* va mostrar que un 93% dels usuaris que havien descarregat una aplicació a l'últim mes estaven disposats a pagar per un joc, mentre que només el 76% estava disposat a pagar per a un altre tipus d'aplicació.

Aquest mercat podria augmentar fins als 13.000 milions de dòlars anuals al 2014, diuen a VentureBeat. I, units als jocs en línea, podrien arribar a dominar el 50% dels ingressos del mercat dels videojocs.

Però no s'ha d'avançar tant en el temps per a observar que aquest és un mercat que pot resultar molt rentable. Recentment Apple va anunciar que ja havia arribat a les 15.000 milions d'aplicacions descarregades, i gràcies a aquestes descàrregues ha pagat 2.500 milions de dòlars a desenvolupadors.

1.1 Motivacions Personals

La meva motivació per a realitzar aquest projecte ha estat poder conèixer el món del desenvolupament dels videojocs per a plataformes mòbils, en aquest cas per a una consola portàtil com és la Nintendo DS.

Actualment, hi han varies llibreries Open Source per a poder programar per a Nintendo DS, però no hi ha motors gràfics i d'esdeveniments Open Source, per a poder realitzar fàcilment videojocs de rol (RPG). D'aquesta manera s'intentarà crear

una eina per a poder facilitar la creació de videojocs tipus RPG i publicar-la a alguna de les comunitats que hi ha d'aplicacions Homebrew per a Nintendo DS.

1.2 Propòsits i Objectius

L'objectiu principal d'aquest PFC és la creació d'un motor gràfic i d'esdeveniments per a jocs RPG per a plataformes mòbils, en aquest cas per a Nintendo DS. S'ha de remarcar que el que s'està proposant no és la programació d'un videojoc, sinó que el que es desenvoluparà és el motor de joc. Aquest motor comptarà no solament amb la funcionalitat bàsica de visualització i animació d'escenaris i personatges, detecció de col·lisions entre personatges i elements de l'escenari, sinó que a més, comptarà amb les funcionalitats pròpies dels jocs més sofisticats (esdeveniments). Aquest motor és compatible per a Nintendo DS i/o qualsevol emulador d'aquesta plataforma per a PC.

1.3 Estructuració de la documentació

Aquesta memòria està estructurada en quinze capítols fonamentals que recullen tota la documentació i explicacions necessàries per a la completa comprensió del projecte.

1.-Introducció, motivació, propòsits i objectius del projecte. En aquest capítol s'explicarà el perquè del desenvolupament d'aquest projecte, quins són els objectius proposats i com s'ha organitzat el desenvolupament.

2.-Estudi de la viabilitat. En aquest capítol es justifiquen els paràmetres que fan possible el desenvolupament del projecte.

3.-Metodologia. Aquest capítol conté l'explicació de la metodologia utilitzada.

4.-Planificació. En aquesta etapa es defineix l'estratègia seguida per arribar als objectius plantejats.

5.-Marc de treball i conceptes previs. En aquest capítol es descriuen els diversos aspectes relacionats amb el desenvolupament general del projecte, que ajudaran a entendre millor els següents capítols. També es tractaran les principals accions desenvolupades durant les primeres etapes de la realització del projecte. S'inclouran els passos d'estudi i aprenentatge de conceptes que s'hagin utilitzat pel desenvolupament.

6.-Requisits del sistema. En aquest capítol es defineixen els requeriments del

software, les quals recullen, a grans trets, els objectius de l'aplicació juntament amb les funcionalitats que es volen obtenir. Aquest document permet entendre els elements que rodegen al sistema informàtic que s'intenta construir.

7.-Estudis i decisions. Aquesta secció conté una descripció de les eines utilitzades, amb les seves característiques i l'ús que se'ls hi ha donat.

8.-Anàlisi i disseny del sistema. Aquest apartat proporciona una comprensió precisa de les necessitats del sistema, és a dir, s'encarrega de la investigació del problema a resoldre, però no s'interessa a trobar una solució. Fent servir l'enginyeria del software, aquesta secció es tradueixen els requeriments nombrats en capítols anteriors a un llenguatge més formal. La part del disseny permet augmentar el nivell d'especificació, y realitzar un esquema d'implementació del sistema mitjançant diverses eines de programació orientada a objectes, explicant les classes i els mètodes que formen el projecte.

9.-Implementació i proves. En aquest capítol es donen a conèixer com s'ha implementat l'aplicació, les classes i els mètodes que resulten més significatius per a la comprensió del funcionament de l'aplicació.

10.-Resultats. En aquest capítol es mostren proves d'execució de l'aplicació, es mostren imatges de tot allò que es pugui visualitzar del conjunt que ha estat implementat.

11.-Conclusions. En aquest apartat s'exposaran les conclusions obtingudes un cop finalitzat el projecte.

12.-Treball futur. En aquesta secció s'exposa tot allò que es podria millorar en l'aplicació, o ampliar-la de forma interessant.

13.-Bibliografia. Aquest capítol conté les referències utilitzades per al desenvolupament del projecte.

14.-Manual d'usuari i instal·lació. Aquesta secció inclou les especificacions del funcionament del producte.

2. Estudi de Viabilitat

Per a desenvolupar el projecte no es requereix d'una gran infraestructura i els costos d'estructura son limitats.

Els materials amb els quals es contava des d'un principi són els següents:

- Ordinador amb sistema operatiu Windows XP.
- Software per a poder programar per a consoles portàtils (**devkitPro**).
- Llibreries especifiques per a Nintendo DS (**PAlib**).
- Consola Nintendo DS i/o emulador per a aquesta plataforma.

En el transcurs del projecte no es va necessitar cap recurs extra, per tant no va haver-hi costos addicionals.

2.1 Recursos Humans

Tot videojoc, per simple que sigui, necessita tres equips bàsics, com són l'equip de programació, l'equip de disseny gràfic i l'equip de so, i en el cas dels jocs de Rol requereixen d'un equip disseny. En aquest cas com que el que s'està programant és un motor gràfic i d'esdeveniments, la figura de l'equip de disseny no és necessària. L'equip de disseny gràfic i so no han estat utilitzats ja que tot els dissenys gràfics i sons s'han obtingut a través de comunitats web. Per tant, el rol que he assumit ha estat el de l'equip de programació.

2.2 Avaluació prèvia de costos i medis

Pel desenvolupament del sistema és necessari portar a terme un estudi de viabilitat tècnica, viabilitat econòmica i viabilitat legal. Degut a que el videojoc presenta riscos baixos i pocs problemes legals, només es consideraran els següents:

- Estudi de viabilitat tecnològica.
- Estudi de viabilitat econòmica.

2.2.1 Estudi de viabilitat tecnològica

L'estudi de la viabilitat tecnològica comenta amb una definició tècnica del motor gràfic i d'esdeveniments proposat, donant res posta a les següents preguntes: Quina tecnologia es requereix per a aconseguir les funcionalitats i els rendiments del motor gràfic i d'esdeveniments? Quins nous materials, modes o processos es requereixen? Com afectaria al cost aquests elements tecnològics?

Afortunadament, ja es disposaven dels dispositius hardware necessaris per poder realitzar el projecte, és a dir, un ordinador capaç de realitzar tota la programació, la interpretació del codi i la visualització dels resultats, i una consola Nintendo DS o emulador per a PC d'aquesta plataforma per a poder realitzar les proves.

En la part de programari, s'ha optat per utilitzar programari lliure i/o gratuït. S'ha optat per el software **devkitPro** per a poder desenvolupar per a consoles portàtils i les llibreries **PAlib** que son especifiques per a Nintendo DS.

2.2.2 Estudi de viabilitat econòmica

La valoració de la viabilitat econòmica s'ha separat en dos parts: els costos dels recursos humans i els costos dels recursos tecnològics.

2.2.2.1 Cost dels recursos humans

Per a calcular els costos dels recursos humans s'ha associat un perfil o rol de treballador per a cada una de les tasques. En aquest projecte només s'han utilitzat 2 perfils ja que la resta de recursos s'han obtingut de recursos externs i no s'han hagut de delegar a cap persona en concret. D'aquesta manera tenim 2 perfils diferents amb els següents costos associats:

- Cost analista/dissenyador: 30 € / hora.
- Cost programador: 25 € / hora.

Tasca	Perfil	Hores	Cost
Estudi entorn **devkitPro** i llibreries **PAlib**	Analista	32	960 €
Disseny algorismes	Analista	32	960 €
Implementació	Programador	120	3000 €
Proves i optimitzacions	Programador	24	600 €
Documentació	Analista	80	2400 €
Total		**288**	**7920 €**

2.2.2.2 Cost dels recursos tecnològics

En quant als costos dels recursos tecnològics s'ha de tenir en compte que els costos associats només son referents a recursos Hardware, ja que tot el programari que s'ha fet servir és programari lliure.

Referent als recursos Hardware no ha suposat cap despesa perquè ja es disposava d'aquests, de tota manera si s'haguessin d'obtenir s'hauria de comptar amb un ordinador i una consola Nintendo DS, on el cost total d'aquests 2 elements seria d'uns 750€.

3. Metodologia

Actualment hi ha moltes metodologies de desenvolupament eficients i diferenciables, des de les més antigues com la metodologia Waterfall fins a les més modernes tècniques Agile. En aquest projecte no s'ha seguit cap metodologia d'un tipus concret, ja sigui Spiral, Scrum o Iterative. S'ha seguit una metodologia específica del projecte **skyline**Engine. S'ha definit un tipus de metodologia que funcionés bé pel projecte tal i com es mostra en el següent esquema (veure Figura 1):

1. Triar el treball a desenvolupar.

2. Decidir el llenguatge de programació i les eines a utilitzar.

3. Aprendre el llenguatge de programació i el funcionament de les eines escollides.

4. Estructurar el treball en parts segons les funcions que ha de realitzar.

5. Desenvolupar la part corresponent seguint l'ordre de l'estructura del treball.

6. Fer comprovacions per tal de confirmar que el funcionament és correcte al finalitzar cada part.

    1. Si al fer les comprovacions el resultat no és l'esperat, es torna al punt 5 per a realitzar els canvis oportuns en la última part desenvolupada o en les anteriors, si és necessari.

    2. Si al fer les comprovacions el resultat és l'esperat, es desenvolupa la part següent tornant al punt 5, en cas que s'hagin finalitzat les parts amb les seves respectives comprovacions s'inicia el punt 7.

7. Unir totes les parts desenvolupades i comprovar que el funcionament és correcte.

    1. Si al fer les comprovacions el resultat no és el resultat esperat, es torna al punt 5 per a realitzar els canvis oportuns en l'ultima part desenvolupada o en les anteriors, si és necessari.

    2. Si al fer les comprovacions el resultat és l'esperat, s'inicia el punt 8.

8. Generar diferents models d'exemple per a comprovar que el funcionament és el correcte.

    1. Si al fer les comprovacions el resultat no és l'esperat, es torna al punt 5 per a realitzar els canvis oportuns en l'ultima part desenvolupada o en les anteriors, si

és necessari.

2. Si al fer les comprovacions el resultat és l'esperat s’inicia el punt 9.

9. Documentar.

Tal i com es pot veure, consisteix en dividir el projecte en mòduls i organitzar en el temps el desenvolupament, el temps de verificació i de correcció. Tant durant el temps de desenvolupament com de verificació es fa un seguiment mitjançant tutories setmanals o bisetmanals depenent de l’etapa, ja que en els inicis la manera d’avançar és molt més lenta que al final i no sempre és necessari fer tutories setmanalment. Quan s’acaba un mòdul, sempre que es pugui, es finalitza totalment de manera que no es torna a tocar, així garantim que els errors que puguin sorgir en el mòdul actual són únicament d’aquest i no de cap dels anteriors, o almenys que afectin el mínim possible.

La tècnica que s’ha seguit per a la creació d’aquesta aplicació és la de disseny descendent. Per això hem utilitzat com a metodologia el llenguatge UML (Llenguatge de Modelatge Unificat o Unified Modeling Language), que tot i no ser una metodologia, és un llenguatge de modelat estàndard pel camp de l’enginyeria del programari. L’UML s’utilitza per definir un sistema, per detallar els seus elements, per documentar i construir. Per aconseguir això, l’UML disposa de nombrosos tipus de diagrames que mostren diversos aspectes dels elements representats.

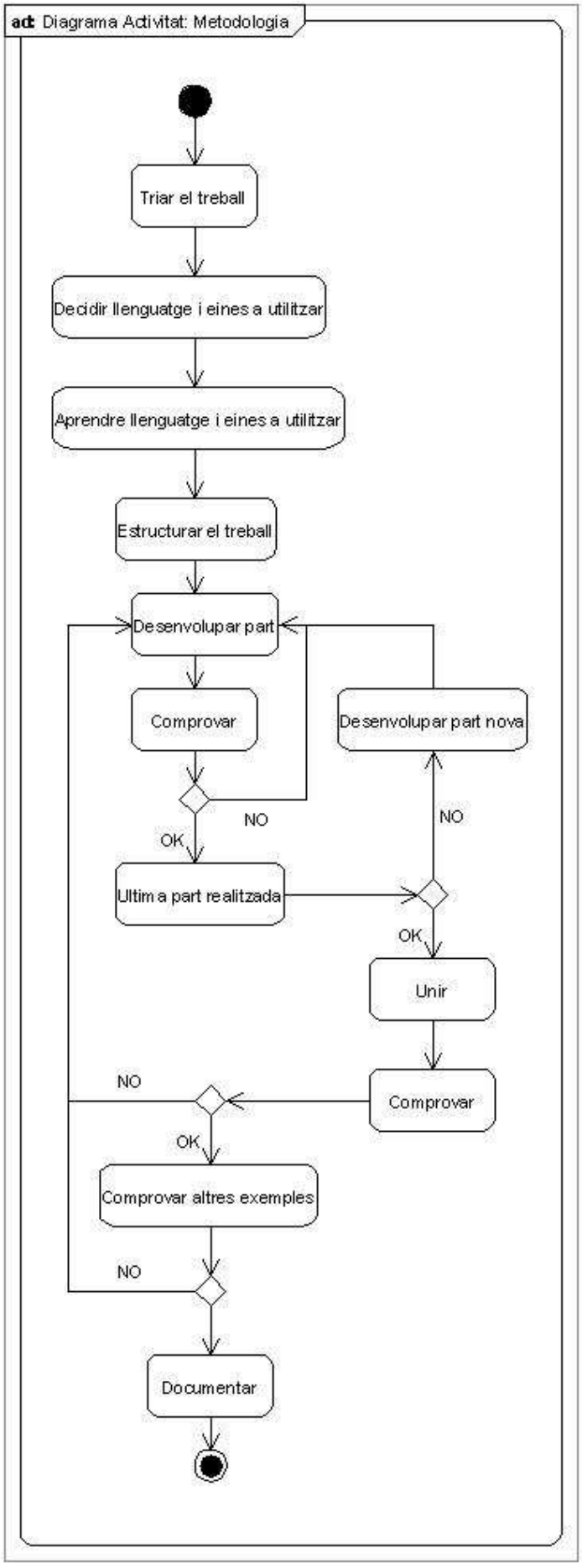


Figura 1: Diagrama d'activitat de la metodologia.

4. Planificació

Aquest projecte es va començar al Juliol del 2011 i inicialment s'havia programat per acabar-lo al Juliol / Setembre 2012, però degut a problemes professionals no es van complir els terminis inicials.

El primer mes va consistir en la investigació i aprenentatge del funcionament de l'entorn de desenvolupament de les **PAlib**. Un cop assolits els coneixements necessaris, es van estudiar les funcionalitats principals de les **PAlib**. Després d'assolir aquest coneixements, es va procedir a implementar una sèrie de proves per a verificar que el comportament d'aquestes llibreries era adient.

A principis de setembre, finalitzada la fase d'aprenentatge, es va començar a definir l'abast del projecte alhora que es començava amb la implementació de la part gràfica, és a dir, cercar els dissenys gràfics i implementar tota la part d'animació i scrolling de l'escena. Un cop acabada i verificada aquesta part es va procedir a passar el següent punt.

A finals d'octubre es va començar a investigar i dissenyar el procés de col·lisions del personatge principal amb les parts estàtiques de l'escena. Un cop finalitzada la tasca d'investigació es va procedir al disseny i la implementació d'aquesta part, que va resultar ser conflictiva, degut a que per a realitzar aquesta part s'ha d'utilitzar un software Open Source específic que no acabava d'adaptar-se al disseny del procés de col·lisions.

A mitjans del mes de març es va iniciar el disseny i implementació del motor d'esdeveniments per a tots els elements (estàtics i dinàmics) de l'escena. Mitjançant el polimorfisme es van dissenyar comportaments específics per a cada cas.

Finalment, els últims mesos han estats dedicats a corregir errors que hagin anat sorgint, millora de codi, arrodonir i organitzar la memòria d'aquest projecte.

El pla de treball es van definir amb 9 tasques:

- Escollir Projecte: Es defineix quin tipus de projecte es realitzarà.
- Planificació inicial del projecte: Es defineix la planificació inicial i una idea inicial de fins on arribarà el projecte.
- Aprenentatge de les eines a utilitzar: Instal·lació de l'entorn de desenvolupament, juntament amb la resta d'eines necessàries per a poder portar a cap el projecte.

Investigació i aprenentatge de les **PAlib**, així com adaptació de les mateixes per a poder desenvolupar tota la part de codi amb Programació Orientada a Objectes.

• Disseny i implementació del control del personatge: Disseny i implementació de tota la part gràfica del projecte així com el control del personatge, l'animació i scrolling.

• Disseny i implementació de col·lisions: Disseny i implementació de les col·lisions del personatge envers els elements estàtics de l'escena.

• Disseny i implementació del motor d'esdeveniments: Disseny i implementació de les estructures de dades necessàries per a poder desenvolupar mitjançant herència i polimorfisme diferents comportaments produïts per diferents esdeveniments.

• Verificació i proves dels algoritmes finals implementats. Comprovació i verificació dels algoritmes implementats i correcció dels possibles errors sorgits.

• Creació d'una demo. Implementació d'una demo fent servir tots els mòduls implementats.

• Documentació. Redactat de la memòria del projecte. Aquesta tasca s'ha desenvolupat al llarg de tot el projecte paral·lelament amb les altres tasques definides.

4.1 Temps estimat

Inicialment es va planejar que el projecte durés 12 mesos però la distribució que es va triar en un primer moment no va estar factible, quedant la distribució final de la següent manera (veure Figura 2).

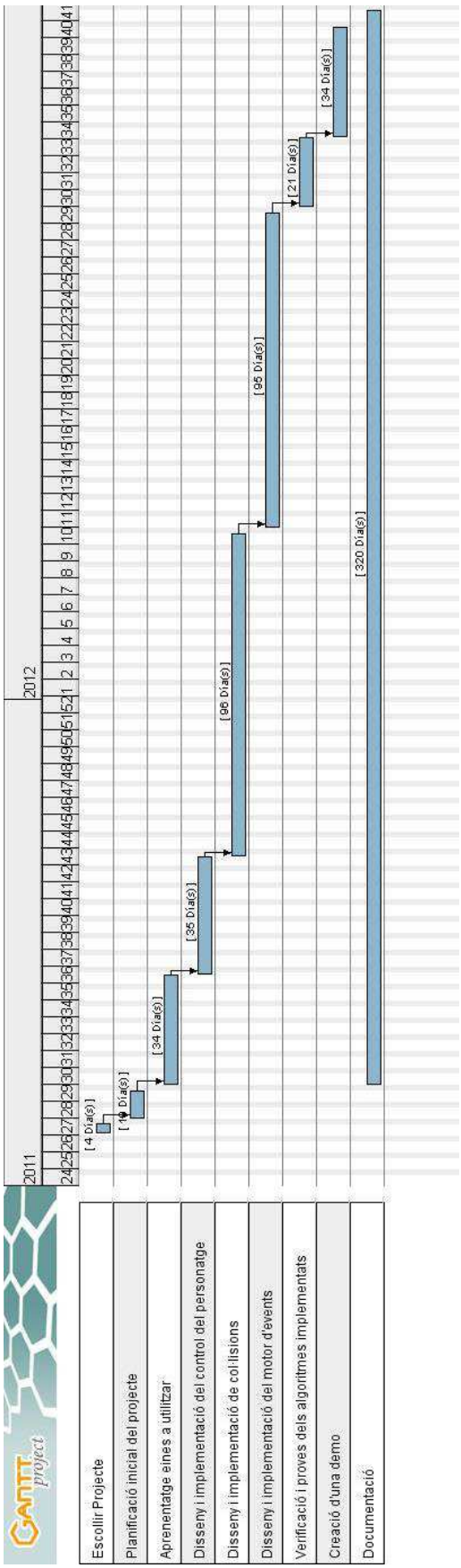


Figura 2: Diagrama de Gantt de la planificació final.

5. Marc de treball i conceptes previs

En aquest apartat seran detallats els conceptes necessaris per tal de poder entendre els algorismes desenvolupats i les eines utilitzades per poder dur a terme amb èxit aquest projecte.

5.1 Nintendo DS

Nintendo DS és la videoconsola portàtil de Nintendo pertanyent a la setena generació de consoles (fins ara, les generacions venien marcades per les consoles de sobretaula). Va sortir al mercat l'any 2004 a Japó i EE. UU. arribant a Europa l'11 de març de 2005. Al gener del 2011, la Nintendo DS es va convertir en la consola més venuda de la història, superant a la Sony PlayStation 2.

Fins a data d'avui, han sortit 4 versions de la consola: DS; DS Lite (reduint la mida); DSi amb millores de hardware i 2 càmeres; i DSi XL, un model més gran que la DSi que inclou pantalles més grans que la seva predecessora (veure Figura 3).

Figura 3: Nintendo DS, Nintendo DS Lite, Nintendo DSi i Nintendo DSi XL. Font: Nintendo.

El disseny de la consola recorda als Game & Watch Multi-Screen de Nintendo dels anys 80, tenint 2 pantalles en un dispositiu "clamshell" (dispositius electrònics que es pleguen mitjançant una frontissa), sent la característica més definitòria de la Nintendo DS el fet que la pantalla inferior sigui tàctil.

Per últim els jocs de la consola venen en el format "Nintendo DS Game Card", un cartutx dissenyat per Nintendo específicament per a la videoconsola. Tenen unes dimensions de 33 mm x 35 mm x 3,8 mm i una capacitat des de 8MB fins a 512MB, a més de una petita quantitat de memòria flash per a poder emmagatzemar partides

guardades i altres dades del jugador. Els models DS i DS Lite tenen a més una ranura compatible amb cartutxos de Game Boy Advance, mentre que els altres dos models de DSi inclouen una ranura per a targetes SD.

En el primer trimestre de 2011 ha sortit a la venta la successora com a videoconsola portàtil de Nintendo, la Nintendo 3DS, amb una pantalla amb més potencia que tota la gamma de DS i la capacitat de reproduir 3D estereoscòpic sense necessitat de utilitzar ulleres.

Aquestes són les característiques tècniques que ofereixen Nintendo DS i Lite:

• Dos pantalles retroiluminades de 3 polzades separades per 21mm, amb una resolució màxima de 256x192 píxels cadascuna, fins 260000 colors (5 bits per canal).

• La pantalla inferior és tàctil i reconeix varies pulsacions simultànies, tornant un únic resultat, el baricentre.

• Dos processadors que poden executar codi simultàniament:

- ARM9: ARM946E-S, la CPU principal, 67 MHz, entre 200 y 300 MIPS, arquitectura RISC de 32 bits.
- ARM7: ARM7TDMI, coprocessador, 33 MHz, sobre 20 MIPS, arquitectura RISC de 16/32 bits.

• 4 Megabytes de RAM integrada que en la major part estan compartits pels dos processadors. A més, 656 KB de RAM per a vídeo i una memòria no volàtil molt limitada per a les preferències d'usuari.

• Una GPU composta de dos sistemes de renderitzat 2D (un per a cada pantalla) i un sistema de renderitzat 3D que pot renderitzar fins a 120000 triangles per segon a una taxa de 60 fps i una taxa de farciment de 30 milions de píxels per segon. La GPU està integrada en el mateix xip que els dos processadors.

• Un pad de direcció i 8 botons (A, B, X, Y en forma de creu; L, R als laterals; i Start i Select.

• Targeta de xarxa integrada, capaç d'oferir el protocol propietari NiFi (Nintendo Wifi) i IEEE 802.11b, per a comunicació entre consoles i Internet respectivament.

• Una sortida d'àudio de 16 canals amb altaveus estèreo i una sortida estàndard per a auriculars.

• Micròfon.

• Dos slots per a cartutxos flash, l'Slot-1 per a cartutxos específics de DS i l'Slot-2 compatible amb cartutxos de GameBoy Advance.

• Bateria li-ion.

• Rellotge integrat de 33 MHz, que s'encarrega de mantenir la hora i la data inclòs quan la consola està apagada.

• Dos LEDs informatius, un sobre l'estat del teclat , l'altre sobre la Wifi.

Les versions DSi presenten les següents diferencies:

• La freqüència de rellotge del processador ARM9 augmenta a 133 MHz.

• La memòria RAM integrada augmenta a 16MB.

• Desapareix l'Slot 2.

• S'afegeix una ranura per a targetes SD.

• S'afegeixen dues càmeres de 0,3 Megapixels, una interna i l'altre externa.

5.2 Història dels videojocs

Ens hem de remuntar a l'any 1947 on Thomas T. Goldsmith i Estley Ray Mann patenten un sistema electrònic que simulava un llançament de míssils contra un objectiu. Aquest projecte no es pot considerar un videojoc ja que no hi havia moviment en pantalla. Més tard apareix el Tres en Ratlla (OXO) d'Alexander Sandy Douglas, considerat un joc gràfic per ordinador, però no un videojoc.

Amb el Tennis for Two de William Higinbotham, representant un camp de tennis i una xarxa mitjançant línies i una pilota, podríem entrar en la discussió de que si aquest va ser el primer videojoc o no. El que queda clar és que aquest joc és considerat bastant més realista que el mític Pong.

El 1961 arriba l'Spacewar! creat per Russell a l'ordinador PDP-1. Després d'unes 200 hores de treball, amb molts problemes durant aquest temps, aquest joc que consisteix en dues naus que s'enfronten a l'espai exterior, és presentat. És considerat com el primer joc d'ordinador de la història, ja que el Tennis for Two només usava circuits electrònics.

Deu anys més tard, Nolan Bushnell (fundador d'Atari), decideix fer la seva pròpia versió de l'Spacewar!. Un cop finalitzat el projecte tindrà força problemes per comercialitzar-lo. El provaria en un campus de la Universitat de Stanford (al bar), i encara que va funcionar bé entre els estudiants, globalment va ser un fracàs.

De tot això, Bushnell va aprendre moltíssim i li va donar les idees bàsiques per començar a gestar la seva veritable idea. Més tard fundaria Atari juntament amb Ted Dabney.

Als anys 70 Ralph Baer al costat d'un equip, havia aconseguit fer funcionar (el 1967) un joc de taula per a dos jugadors.

Però, encara, el problema estava a l'hora de comercialitzar el projecte. Va preguntar a moltes empreses i sempre va obtenir respostes negatives per part seva, fins que va trobar a Magnavox (on treballava Bill Enders, el qual els va parlar del projecte de Baer). Finalment aquesta empresa sí que accepta les condicions i comença a treballar per comercialitzar-la.

El 1972 es va llançar la Magnavox Odyssey al mercat, amb 12 jocs diferents. Aquesta consola només podia funcionar en els televisors de la mateixa marca.

Quan Bushnell assisteix a la Magnavox Profit Caravan d'aquest mateix any i veu la consola d'aquesta companyia, té la genial idea de crear un joc similar. Contracten a Allan Alcorn i li encarreguen el projecte. Tres mesos més tard ja té un prototip operatiu i a Bushnell i Dabney els encanta. El bategen com Pong (veure Figura 4).

Figura 4: Pong en una sala de videojocs.

Introdueixen aquest nou joc en bars i de seguida causa una gran sensació, arribant-se fins a "espatllar" les màquines pel fet que les caixes de les monedes estaven a vessar. Pong seria el major èxit mai vist en la indústria dels videojocs fins a aquest moment. No obstant això, Bushnell tindria problemes amb Baer, ja que aquest últim l'acusava de plagi del seu joc. Finalment aconsegueixen un tracte del qual Bushnell surt completament beneficiat.

A partir de l'èxit de Pong, Atari comença a créixer d'una manera espectacular, amb les seves fàbriques treballant sense parar 16 hores seguides creant mobles de Pong. Això va portar a Atari a treure nous jocs al mercat com Space Race, Pong doubles i Gotcha. No obstant això, cap d'aquests va arribar a obtenir l'èxit de Pong.

Moltes empreses comencen a copiar les idees a Bushnell i la competència creix. Atari llavors es posa les piles i comença a treure jocs cada mes. Fins i tot Bushnell ha de crear una empresa d'amagat (KeeGames) per fer veure al consumidor que hi havia una competència real i de passada col·locar un major percentatge de màquines a les sales recreatives.

5.3 Videojocs RPG (Role-playing game)

"Joc de rol" és la traducció usual en català de Role-playing game, literalment: «joc d'interpretació de papers». El videojoc de rol com a gènere de videojocs inclou una àmplia varietat de sistemes i estils de joc. Alguns elements fortament associats als jocs de rol, com el desenvolupament estadístic de personatges, han estat adaptats àmpliament a altres gèneres de videojocs. Encara que usin la paraula rol, no poden ser considerats com jocs de rol en si. La proliferació d'aquests tipus de jocs en els últims anys i l'ús de la paraula rol per nomenar-los ha fet que un gran nombre de persones cregui (erròniament) que els videojocs de rol són realment jocs de rol en si mateixos.

Els principals atractius que persegueixen els fanàtics d'aquesta classe de videojoc són:

- La jugabilitat del combat, cada vegada més complex i realista en certs aspectes.
- Els moviments i trets humans més naturals en general.
- Els béns virtuals que es posseeixen (especialment armes i efectes de guerra encantats o màgics, que faciliten el joc notablement, o que simplement, en

presumeixen)

• El detall en les estadístiques que llança l'aventura.

• Els reconeixements al temps invertit en el videojoc (nivells de les habilitats assolits, que defineixen la respectabilitat del jugador davant els altres aficionats).

La durada d'un sol joc d'aquesta classe demanda més temps mitjà que la resta dels jocs a causa, entre d'altres coses, a l'extensió dels mapes i a la varietat de personatges i missions secundàries disponibles durant l'aventura, que els atorga una major profunditat, així com les converses més o menys extenses i freqüents amb diferents NPC (Non-Player Character) per poder completar les missions, donant-li un desenvolupament molt menys directe que el d'altres gèneres.

Ara es definirem els diferents tipus de jocs dins del gènere dels jocs de rol. Són generalitzacions i no han de ser definicions exactes.

• RPG (Role-playing Game): Conté a tots els jocs de ROL, però s'usa principalment per als jocs occidentals, que es caracteritzen per arguments foscos i majoritàriament medievals.

• JRPG (Japanesse Role-playing Game): Són els jocs de ROL creats al Japó, i que es distingeixen pels seus protagonistes d'estil Visual kei, així com pels seus arguments menys foscos i ambientacions fantàstic-futurista.

• ARPG (Action Role Playing Game): Aquest tipus de ROL pot contenir als dos anteriors, i es caracteritza per usar combat directe sense torns, més semblant a qualsevol joc d'acció que al clàssic plantejament "rolero" de combat per torns heretat dels taulers. La resta de components com l'evolució, personalització, trama, etc, no es veuen afectats per aquest matís.

• TRPG (Tactic Role playing Game): Jocs de ROL tàctics, en què l'avanç es realitza al llarg d'escenaris dividits en quadrícules i on els atacs es realitzen per torns, tenint molt pes l'estratègia en les batalles.

• MMORPG (Massive multiplayer online Role-playing game): És un gènere de videojoc de ROL on participen un gran nombre de jugadors interactuant entre si en un món obert. En aquest tipus de jocs hi ha l'opció de xatejar amb la resta de jugadors per organitzar les quest (missions) o comerciar, entre altres coses, i és habitual crear clans.

5.4 El motor de joc

En el desenvolupament de videojocs normalment s'utilitza un motor de jocs (Game Engine) degudament adaptat a les necessitats dels desenvolupadors, de manera que no hagin de programar directament contra les llibreries gràfiques. També solen implementar motors de físiques, amb suports per a models gràfics i música.

El motor de videojocs és aquell que des d'un baix nivell s'ocupa d'administrar tot allò que es pot veure, com a models i textures, amb una combinació d'eines i funcions que permeten programar a un nivell més alt.

Hi ha diversos motius pels quals l'ús d'un bon motor gràfic és important. Alguns d'ells són els següents:

- Facilita el desenvolupament de l'aplicació.
- Obre noves oportunitats de negoci, com podria ser la venda de llicències del motor per a la realització d'altres jocs.
- Aconsegueix l'abstracció de la plataforma, és a dir, que si un motor es pot executar en diverses plataformes, la nostra aplicació també podrà.
- La separació del motor i els continguts permet tenir diversos grups de treball en paral·lel.
- Proporciona beneficis a tercers, ja que tots els avenços que es facin en el motor podran beneficiar a diverses aplicacions basades en el motor de joc.
- Permet emfatitzar la part artística, ja que en tenir més grau d'abstracció, es disminueix la càrrega de treball referent a la programació.
- Assoleix major modulació i reaprofitament del codi.

La part que sol tenir més pes en un sistema que utilitza un motor de videojocs sol ser la part del motor de renderització, seguit del graf d'escenes i el detector de col·lisions.

6. Requisits del sistema

En aquest capítol es descriuen els requeriments, els quals expliquen a grans trets els objectius de l'aplicació, juntament amb les funcionalitats desitjades. Dins de tota aplicació apareixen bàsicament dos tipus de requeriments:

- Requeriments funcionals: Descriuen quins són els serveis que oferirà l'aplicació, independentment de la implementació.
- Requeriments no funcionals: Informen sobre les restriccions que vénen imposades pel client o pel propi problema.

A continuació es descriuen en detall els dos tipus de requeriments comentats.

6.1 Requeriments funcionals

En aquest apartat es descriuen els serveis que oferirà aquesta aplicació, sense tenir en compte la seva implementació.

S'entenen com requeriments generals del sistema les principals funcionalitats del programari a desenvolupar, reflectint en tot moment les responsabilitats del motor de joc desenvolupat.

A continuació es mostren els principals requeriments:

- Iniciar una partida.
- Moure el personatge per l'escenari.
- Col·lidir amb elements de l'escenari.
- Interactuar amb elements de l'escenari.

L'usuari podrà executar el joc i es carregarà una pantalla d'inici amb varies opcions a triar (veure Figura 5). Al iniciar una partida es carreguen en memòria totes les llibreries i paràmetres necessaris per a fer funcional el motor de joc. Un cop iniciada la partida es farà una càrrega en pantalla de tots els elements necessaris.

Figura 5: Exemple de pantalla d'inici d'un videojoc de la Nintendo DS.

Un cop carregat el motor de jocs i tots els elements, l'usuari podrà moure el personatge principal mitjançant el pad direccional. A la Figura 6 podem veure el moviment del personatge principal en un joc 2D un cop s'ha pressionat la direcció en el pad direccional.

Figura 6: Exemple de moviment provocat al pressionar el pad direccional.

El personatge principal pot col·lidir amb diferents elements de l'escena, és a dir, si el personatge principal es topa amb un element o amb un límit del mapa, aquest no deixarà continuar el desplaçament (veure Figura 7).

Figura 7: Exemple de col·lisió provocat al pressionar el pad direccional.

La interacció amb elements de l'escena pot ser amb elements estàtics (ítems o NPC's) o dinàmics (enemics). L'usuari premerà una tecla i depenent de l'esdeveniment associat a l'ítem farà una acció o una altra. En la següent figura es pot veure l'evolució d'un esdeveniment associat a una element de l'escena.

Figura 8: Exemple d'interacció amb elements de l'escena.

6.2 Requeriments no funcionals

En aquest apartat es descriuen aspectes sobre com ha de ser l'aplicació i no sobre que ha de fer. En tot projecte cal prestar especial atenció a tots els aspectes que s'han de tenir en compte quan s'ha de dissenyar un sistema, més enllà de l'explicació funcional detallada presentada anteriorment. Els requeriments no funcionals del sistema són aquells que fan referència a restriccions del tipus de disponibilitat de recursos, seguretat o interfícies externes (hardware i software), entre altres. Aquestes condicions permetran executar l'aplicació sense problema.

S'han definit els següents requeriments no funcionals:

- El motor de joc ha de ser eficient: no fer càlculs redundants ni utilitzar més

memòria de la necessària.

• També hauria de ser extensible, permetent incorporar noves variants d'algorismes a mesura que es vagin desenvolupant.

• La interfície ha de ser intuïtiva i usable.

• S'ha de fer una documentació sobre el funcionament bàsic del programa.

• Hauria de poder funcionar en diversos sistemes operatius.

En quant a l'aplicació implementada, cal dir que des del punt de vista de la seguretat, no hi ha cap requisit de control d'accés al programari, ja que es tracta d'una aplicació on els usuaris que accedeixen només tindran un rol. Tampoc disposa de dades confidencials o d'alt risc, pel que no té sentit un sistema de protecció de dades.

Com que la plataforma on s'ha desenvolupat el projecte és **devkitPro**, els requeriments mínims són els de **devkitPro**, el qual és un software multi-plataforma ja que es pot executar per GNU/Linux , MS Windows o Mac OS X, i en el apartat de hardware 512 MBytes de RAM, processador de 32 bits d'AMD o Intel, 1GiB de disc dur, i una rata de 3 botons.

Cal destacar que les llibreries que s'utilitzen en el projecte, **PAlib** compleixen els requeriments, ja que són multi-plataforma i són compatibles amb C / C++ que incorpora **devkitPro**. A més a més la llibreria **PAlib**, al ser escrita totalment en C, no fa falta instal·lar-la ja que s'ha incorporat al projecte i funciona per qualsevol sistema operatiu.

La implementació i les proves de l'aplicació s'ha fet amb un equip de les següents característiques:

• Processador Intel Pentium M 740

• 1 Gbytes de RAM

• Sistema operatiu Windows XP 32 bits

• Targeta gràfica ATI Raedon

L'aplicació està programada amb C++.

7. Estudis i decisions

En aquest capítol s'anomenen les llibreries i programes utilitzats per a realitzar aquest projecte. Es mostren les eines que han permès la implementació de l'aplicació, des de les mes bàsiques fins a aquelles que han servit de suport, així com els programes necessaris per a construir la documentació corresponent.

7.1 Sistema Operatiu

Per a desenvolupar aquest projecte s'ha utilitzat el sistema operatiu Windows XP en la versió Home Edition, bàsicament, perquè es disposava d'un equip portàtil amb aquest sistema operatiu, però es podia haver desenvolupat amb GNU/Linux, ja que tot el software utilitzat és Open Source i multiplataforma.

7.2 Software utilitzat

En aquest apartat s'explica cada programa de software utilitzat per a la realització del projecte, així com el paper que exerceix.

7.2.1 DevkitPro

DevkitPro és un paquet d'eines, Open Source i multiplataforma, per al desenvolupament de Homebrew. En l'actualitat està disponible per a GameBoy Advance, Nintendo DS, GP32, Playstation Portable (PSP), GameCube y Wii. El paquet **devkitPro** porta incorporats els següents paquets:

- **devkitARM**: Kit de desenvolupament per a màquines basades en arquitectura ARM.
- **devkitPPC**: Kit de desenvolupament per a màquines basades en arquitectura PowerPC.

- **devkitPSP**: Kit de desenvolupament per a Playstation Portable (PSP).

D'aquests tres paquets només necessitem el **devkitARM**, que permet la compilació de binaris compatibles amb l'arquitectura ARM. El compilador de **devkitARM** fa servir GCC (GNU Compiler Collection), que és un compilador integrat del projecte GNU per C, C++, Objective C i Fortran. La implementació de l'aplicació per Nintendo DS està programada en C++.

DevkitPro porta incorporades les llibreries **libnds** que són les llibreries que adapten el codi C/C++ que realitza el hardware específic de la Nintendo DS. A més la llibreria **libnds**, també porta altres llibreries com són llibreries per a funcions amb la Wi-Fi, per a tractar amb sistemes de fitxers i llibreries per a treballar amb àudio i vídeo.

La decisió de programar amb aquest kit de desenvolupament i no un oficial bàsicament ha estat una qüestió econòmica, ja que per a poder desenvolupar amb un SDK oficial s'ha de ser desenvolupador oficial de Nintendo i pagar el preu de la llicencia de l'SDK, que per aquest projecte és totalment inviable.

7.2.2 Llibreries PAlib

PAlib és un conjunt de llibreries basades en **libnds** i programades en C/C++ que pretenen d'una manera fàcil poder programar per a Nintendo DS. **PAlib** inclou totes les funcionalitats possibles a un alt nivell i amb les següents característiques:

- Suport per a backgrounds de 256 colors.
- Suport per a sprites de 256 colors.
- Motor de text.
- Suport bàsic per a la càrrega d'arxius RAW.
- Suport per a mapes de col·lisions.
- Suport per a sistema de fitxers.
- Implementació de les principals funcions matemàtiques i trigonomètriques.

L'ús d'aquestes llibreries s'ha basat en poder programar a un nivell alt i no haver d'aprofundir en la programació a més baix nivell amb **libnds.**

La implementació de l'aplicació està centrada en les funcions que ofereix la llibreria **PAlib** i el llenguatge C++. Per a saber l'estructura inicial d'un projecte **PAlib** i com organitzar els fitxers del projecte en diferents directoris, hi ha plantilles que faciliten la feina. Aquestes estan a *C:\devkitPro\PAlibTemplate* (si hem fet una instal·lació estàndard de **devkitPro**).

Una plantilla defineix la següent estructura de directoris, (on només es obligatori el directori source):

- source: per a els fitxers font (vàlid per a les extensions .c i .cpp també s'accepten els fitxers capçalera .h).
- include: per a els fitxers capçalera (.h).
- data: per a els fitxers de dades del projecte (s'incorporaran al fitxer .nds).
- music: fitxers de música (.mod, .s3m, .xm o .it) i efectes de so (.wav).
- build: per a els fitxers resultants de la compilació dels fitxers font.

A més de l'estructura de directoris, al directori arrel del projecte també s'inclouen els següents arxius:

- Makefile: fitxer que conté totes les instruccions de compilació. També inclou variables editables per a poder assignar un nom a l'aplicació i una descripció, que apareixerà en el menú de càrrega de la Nintendo DS.
- source/main.cpp: conté el programa principal, aquest ha de tenir una estructura determinada (veure Figura 9).
- logo.bmp: logo de l'aplicació, aquesta es mostrarà en el menú de càrrega de la Nintendo DS.
- Arxiu .nds: arxiu executable resultant de l'execució de Makefile.
- Arxius .bat: arxius executables per lots lligats a Makefile que eliminen executables antics i munten els executables nous.
- Arxius .vhm: també s'inclouen una sèrie d'arxius per a l'edició del projecte amb el programari VisualHam, també inclòs en l'estructura del projecte **PAlib**, però no s'han utilitzat ja que s'ha triat Notepad++ com a IDE del projecte.

L'estructura del main.cpp d'un projecte **PAlib** ha de tenir sempre la següent estructura:

```
1  // Includes
2  #include <PA9.h>       // Include for PA_Lib
3
4  // Function: main()
5  int main()
6  {
7      PA_Init();    // Initializes PA_Lib
8      PA_InitVBL(); // Initializes a standard VBL
9
10     // Infinite loop to keep the program running
11     while (1)
12     {
13         //Insert code here
14         PA_WaitForVBL();
15     }
16
17     return 0;
18 } // End of main()
19
```

Figura 9: Estructura del codi del main.cpp d'un projecte **PAlib**.

Com podem veure en la Figura 9 anterior, l'estructura mínima i obligatòria per a un projecte **PAlib** s'estructura de la següent manera:

- #include <PA9.h>: s'inclou la llibreria **PAlib**.
- void PA_Init(): S'inicialitzen totes les variables i estructures de dades necessàries per a la utilització de les **PAlib**.
- void PA_InitVBL: Inicia les rutines necessàries per a que la Nintendo DS realitzi tasques amb regularitat. Com per exemple, obtenir la posició de l'Stylus.
- While (1): Inicia un dimoni per a que l'aplicació s'estigui contínuament executant.
- void PA_WaitForVBL(): Funció que serveix per a sincronitzar el joc amb la velocitat de refresc de la pantalla (per defecte 60 FPS).

7.2.2.1 Detalls C/C++ per a la Nintendo DS

Per al desenvolupament del projecte s'ha triat el llenguatge de programació C++.

Aquesta decisió té molts avantatges (per exemple la possibilitat de POO amb tot el que implica) però s'ha d'anar en compte al utilitzar la llibreria estàndard de C++ en la Nintendo DS per les STL, les excepcions, els streams i les sobrecarregues de memòria entre d'altres, ja que incrementa molt l'ús de memòria. Per tant s'utilitzarà en la mesura del possible el llenguatge C++ combinat amb les **PAlib**.

La principal diferència que trobem a l'hora de programar amb **PAlib** és que s'utilitzen sinònims per a referir-nos als tipus primitius de dades. Això és una característica heretada de **libnds**. Seguidament s'adjunta una taula per a mostrar aquests canvis:

	Unsigned	**Signed**
Char (8 bits)	u8	s8
Short (16 bits)	u16	s16
Int, float (32 bits)	u32	s32
Double (64 bits)	u64	s64

7.2.2.2 Sortida de text

Amb **PAlib** la sortida de text és una tasca senzilla i disposa de varies funcions per a mostrar text en qualsevol pantalla. El primer que s'ha de fer és inicialitzar el sistema de text mitjançant la instrucció:

• **void PA_InitText(u8 Screen, u8 Bg_select);**

Screen: Atribut que informa de la pantalla on es mostrarà el text (0 per la pantalla superior i 1 per a la inferior).

Bg_select: Atribut que informa de la capa on volem que aparegui el text (valors compresos entre 0 i 3, essent 0 la capa més superior, 1 per a la següent ... i 3 per a la més inferior).

Un cop inicialitzat el text, **PAlib** disposa de 2 funcions per a mostrar text en pantalla:

• **u8 PA_OutputSimpleText(u8 screen, u16 x, u16 y, const char *text);**

• **void PA_OutputText(u8 screen, u16 x, u16 y, char *text,...);**

Screen: Atribut que informa de la pantalla on es mostrarà el text (1 per la pantalla

superior i 0 per a la inferior).

X, Y: coordenades, en tiles (0-31) y (0-23), respectivament (veure Figura 16 pàgina 35).

text: text en ASCIIZ.

També disposem d'una funció per a esborrar tot el text mostrat a la pantalla:

• **void PA_ClearTextBg(u8 screen);**

Un dels avantatges de **PAlib** és que es pot escriure text en les coordenades que el programador vulgui. Amb això s'ha d'anar amb compte a causa que, si s'escriuen dues sortides de text diferents en les mateixes coordenades, es sobreescriuran.

A continuació a la Figura 10 podem veure un exemple de impressió per pantalla d'un text mitjançant les dues funcions.

```
1  // Includes
2  #include <PA9.h>       // Include for PA_Lib
3
4  // Function: main()
5  int main()
6  {
7      PA_Init();    // Initializes PA_Lib
8      PA_InitVBL(); // Initializes a standard VBL
9      PA_InitText(0,0);
10     PA_InitText(1,0);
11
12     s32 x = 0;
13
14     // Infinite loop to keep the program running
15     while (1)
16     {
17         //Insert code here
18         PA_OutputSimpleText(1, 0, 0,"Hello World! Pantalla 1");
19         PA_OutputText(0,0,0,"Hello World! Pantalla %d", x);
20
21         PA_ClearTextBg(1);
22
23         PA_WaitForVBL();
24     }
25
26     return 0;
27 } // End of main()
```

Figura 10: Exemple de sortida de text mitjançant 2 funcions diferents.

A la següent figura podem veure l'execució del codi anterior:

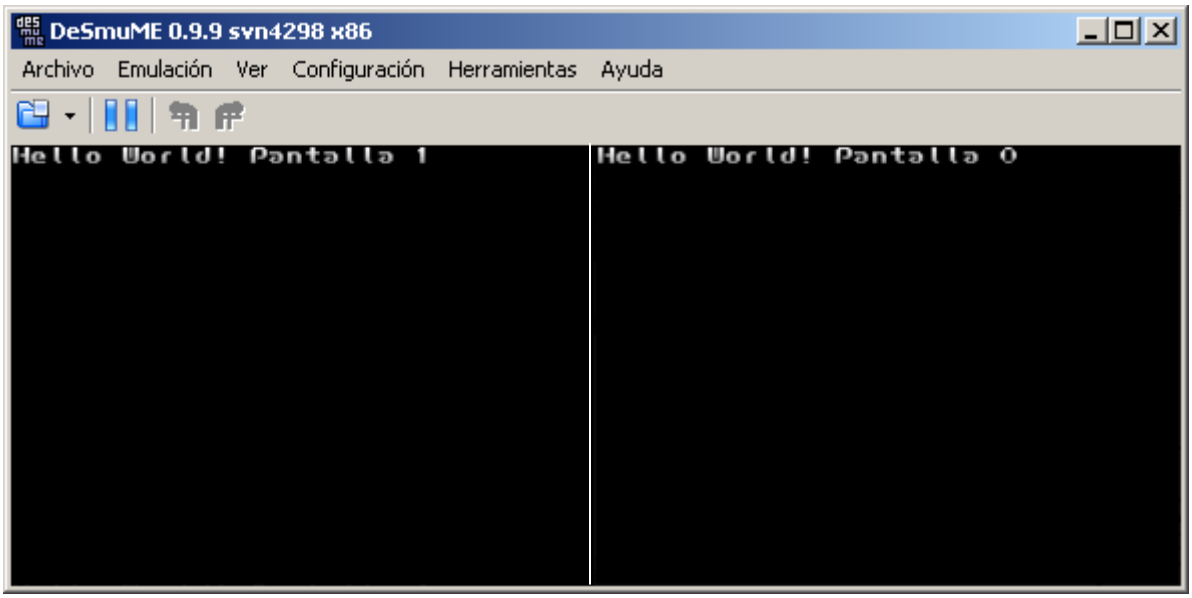


Figura 11: Execució del codi de la Figura 10 en un emulador de Nintendo DS per a PC.

7.2.2.3 Dispositius d'entrada

És molt important saber utilitzar els dispositius d'entrada que ofereix la Nintendo DS per a poder interactuar amb aquesta, ja sigui mitjançant la botonera, la pantalla tàctil, el micròfon o inclòs altres dispositius connectats a l'Slot 2 com a expansió, sense tenir en compte la càmera de la Nintendo DSi.

7.2.2.3.1 Botonera

Per a l'accés a la botonera de la Nintendo DS amb **PAlib**, s'utilitza l'estructura *PAD* que s'actualitza a cada frame. L'estructura és la següent *PAD* + esdeveniment + tecla (veure Figura 12).

Figura 12: Estructura esdeveniments classe *PAD*.

El funcionament és senzill, l'usuari prem una tecla i el sistema avalua quin esdeveniment s'està produint i per a quina tecla s'està produint. Els esdeveniments tenen els següents comportaments:

- Newpress: S'activa quan es prem una tecla. Està actiu durant 1 frame.
- Released: S'activa quan es deixa de prémer una tecla. Està actiu durant 1 frame.
- Held: S'activa quan es manté polsada una tecla. Està actiu mentre es mantingui polsada una tecla.

Aquests esdeveniments retornen 1 o 0 depenent si estan actius o inactius, respectivament. Seguidament es pot veure un exemple de funcionament d'aquests esdeveniments (veure Figura 13).

```
1  // Includes
2  #include <PA9.h>       // Include for PA_Lib
3
4  // Function: main()
5  int main()
6  {
7      PA_Init();    // Initializes PA_Lib
8      PA_InitVBL(); // Initializes a standard VBL
9      PA_InitText(0,0);
10     PA_InitText(1,0);
11
12     // Infinite loop to keep the program running
13     while (1)
14     {
15         //Insert code here
16         if(Pad.Newpress.A){
17             PA_OutputText(0,0,0,"Has polsat A");
18         }
19         else if(Pad.Released.A){
20             PA_OutputText(0,0,1,"Has deixat de polsar A");
21         }
22         else if(Pad.Held.A){
23             PA_OutputText(0,0,2,"Estàs polsant A");
24         }
25         PA_WaitForVBL();
26     }
27
28     return 0;
29 } // End of main()
```

Figura 13: Exemple d'esdeveniments de la classe *PAD*.

7.2.2.3.2 Stylus

Una de les característiques més diferencials de la Nintendo DS és la pantalla inferior que és tàctil. La funcionalitat es semblant a la de la botonera, és a dir, l'usuari fa servir l'*Stylus* i el sistema avalua quin esdeveniment s'està produint. Per a utilitzar les funcions de la pantalla tàctil s'utilitza l'estructura Stylus que està composada per *Stylus* + Esdeveniment, com es pot veure a la Figura 14:

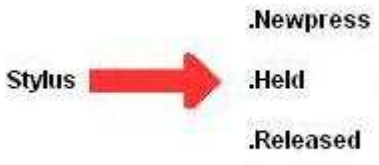


Figura 14: Estructura esdeveniments classe *Stylus*.

Els esdeveniments relacionats amb la classe *Stylus* són els mateixos que els de la classe *Pad* que s'han explicat anteriorment. A més a més podem saber les coordenades de pulsació de l'Stylus, utilitzant les funcions:

- Stylus.X: Retorna la posició X (en píxels).
- Stylus.Y: Retorna la posició Y (en píxels).

En la següent figura es pot veure un exemple d'utilització de les funcions explicades, referents a la classe Stylus:

```
1  // Includes
2  #include <PA9.h>       // Include for PA_Lib
3
4  // Function: main()
5  int main()
6  {
7      PA_Init();    // Initializes PA_Lib
8      PA_InitVBL(); // Initializes a standard VBL
9      PA_InitText(0,0);
10     PA_InitText(1,0);
11
12     // Infinite loop to keep the program running
13     while (1)
14     {
15         //Insert code here
16         if(Stylus.Newpress){
17             PA_OutputText(0,0,0,"Stylus polsat a: (%d,%d)"Stylus.X,Stylus.Y);
18         }
19         PA_WaitForVBL();
20     }
21
22     return 0;
23 } // End of main()
```

Figura 15: Exemple d'esdeveniments de la classe Stylus.

7.2.2.4 Gràfics

La Nintendo DS està especialment dissenyada per a mostrar gràfics a les pantalles. Si bé els 4Mb de memòria principal i 656Kb de memòria gràfica no permeten mostrar gràfics d'última generació, l' arquitectura facilita l'accés a la memòria gràfica sense passar per la CPU, amb el que realitza el procés de dibuixat de forma eficient.

S'ha de tenir en compte les dimensions de les pantalles i la distribució és en tiles. Les pantalles tenen 768 tiles (32 ample x 24 alt) i una resolució de 256 x 192 píxels (veure Figura 16).

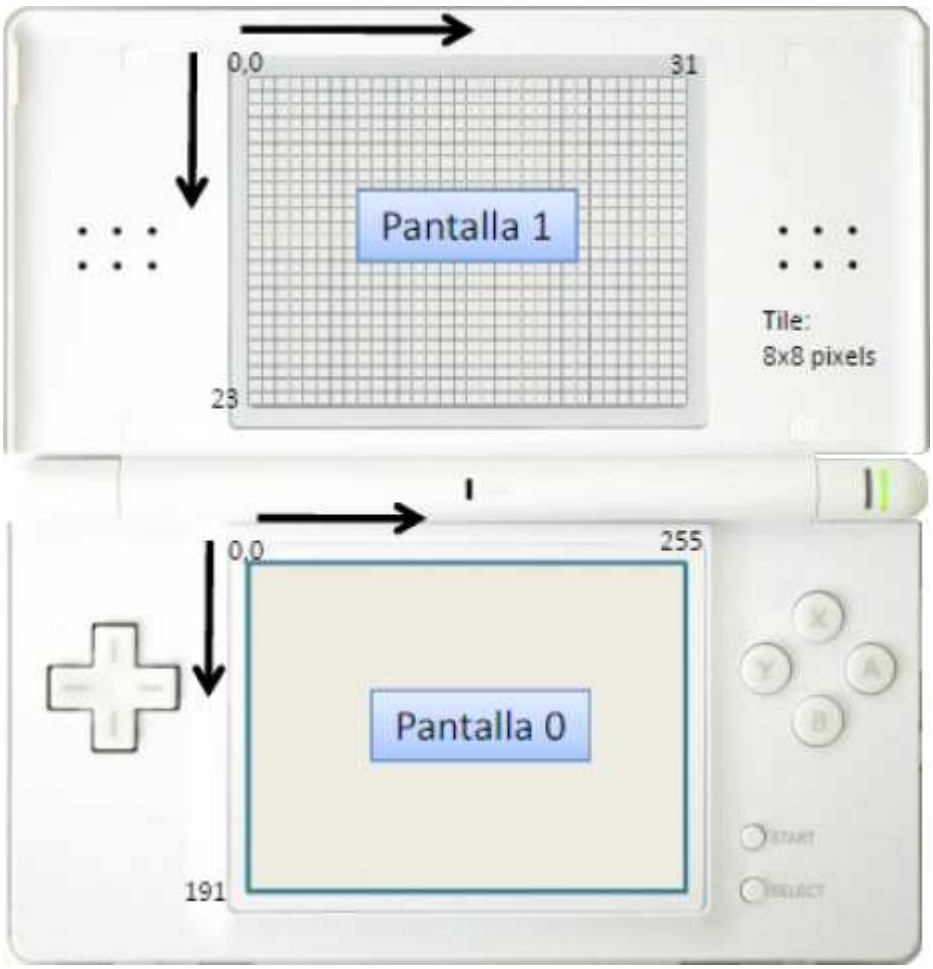


Figura 16: Distribució de la pantalla de la Nintendo DS en tiles (pantalla superior) i píxels (pantalla inferior).

Hi ha dos tipus d'imatges 2D en la Nintendo DS: sprites i backgrounds (fons). La consola pot mostrar fins a 4 backgrounds o 128 sprites per pantalla. També s'ha de tenir en compte que **PAlib** no pot fer servir format d'imatges (.jpg, .png, .jpeg, ...), sinó que primerament se li ha de fer un tractament previ amb el software *PAGfx* per a convertir-les en format de la Nintendo DS (veure apartat 7.2.3).

7.2.2.4.1 Sprites

Els Sprites són imatges 2D, habitualment petites i parcialment transparents, utilitzades en programació gràfica (especialment en videojocs) per a representar pràcticament tots els objectes que hi ha en pantalla.

Els sprites es poden traslladar per la pantalla, es poden rotar, escalar o inclòs animar actualitzant la seva imatge mostrada.

En la Nintendo DS, els sprites presenten 3 modes de color:

• Paletes de 16 colors, amb un total de 16 paletes diferents per pantalla. Aquest mode es feia servir en Game Boy Advance.

• Paletes de 256 colors, amb un total de 16 paletes diferents per pantalla. És el que se sòl utilitzar.

• Sprites de 16 bits sense utilitzar paleta. No se solen fer servir ja que consumeixen molts recursos.

També s'ha de tenir en compte la mida dels sprites, ja que no poden tenir la mida que nosaltres vulguem. Han de seguir les limitacions de la següent taula de mides (en píxels):

	8	16	32	64
8	8x8	8x16	8x32	
16	16x8	16x16	16x32	
32	32x8	32x16	32x32	32x64
64			32x64	64x64

De manera que si tinguéssim un sprite de 42x42 píxels hauríem de fer servir la mida 64x64 píxels i utilitzar el color de transparència (Magenta) per a emplenar l'espai sobrant. En la següent figura podem veure un exemple d'sprite:

Figura 17: Exemple d'sprite amb elements dividit en 4 frames (0-3) de mida 16 x 16.

7.2.2.4.1.1 Inicialització i càrrega

Per a la inicialització dels sprites es fan servir dos funcions diferents, una per carregar la paleta i l'altre per a crear l'sprite:

• **void PA_LoadSpritePal (u8 screen, u8 palette_number, void * palette);**

• **void PA_CreateSprite(u8 screen, u8 obj_number, void *obj_data, u8 obj_size, u8 color_mode, u8 palette_number, s16 x, s16 y);**

Screen: Atribut que informa de la pantalla on es mostrarà el text (1 per la pantalla superior i 0 per a la inferior).

palette_number: Número de paleta (0-15).

Palette: Nom de la paleta amb el següent format **(void *)** Nom_paleta.

obj_number: Número d'sprite que farem servir (0-127 per a cada pantalla).

obj_data: Nom de l'sprite a carregar amb el següent format **(void *)** Nom_Sprite.

obj_size: Mida de l'sprite amb el següent format (**OBJ_SIZE**_*MIDAX***X***MIDAY*), essent *MIDAX* i *MIDAY* les mides en píxels de l'sprite, fent servir les mides compatibles de la taula vista anteriorment.

color_mode: Mode de color 256 o 16 bits (1 o 0 respectivament).

X, Y: coordenades (respecte la pantalla) x i y en píxels, respectivament.

La primera funció carrega la paleta en memòria, la segona crea i carrega l'sprite en la posició (x,y) de la pantalla.

Per a esborrar l'sprite es fa servir una funció d'esborrat, el que fa és eliminar l'sprite de memòria i la seva representació gràfica en la pantalla.

• **void PA_DeleteSprite (u8 screen, u8 obj_number);**

Un cop esborrat l'sprite es descarrega de la memòria i s'elimina de la pantalla, però la paleta encara està carregada en memòria. Per tant, si volguéssim tornar a mostrar l'element hauríem de tornar a crear l'sprite i no caldria fer la càrrega de la paleta. El desavantatge que tenim amb aquest mètode de mostrat - esborrat, és que a cada creació d'sprite haurem de tenir el nom de l'sprite i és un mètode totalment inviable si tenim molts elements en pantalla. Per a resoldre aquest problema es fan servir les següents funcions com a alternativa de la creació anterior d'sprites:

• **u8 PA_CreateGfx((u8 screen, void *obj_data, u8 obj_size, u8 color_mode);**

• **void PA_CreateSpriteFromGfx (u8 screen, u8 obj_number, u16 obj_gfx, u8**

obj_size, u8 color_mode, u8 palette, s16 x, s16 y);

La primera funció carrega l'sprite en memòria i retorna la posició de memòria on es guarda tota la informació referent a l'sprite. La segona funció crea l'sprite a partir de la posició de memòria retornada anteriorment. Aquesta manera de treballar és més senzilla, ja que podem guardar l'adreça de memòria d'una manera més eficient que no pas guardant el nom de cada sprite.

7.2.2.4.1.2 Animació

L'animació dels sprites ve definida per 3 funcions bàsiques que assignen el frame a mostrar, inicien l'animació des de un frame origen fins a un frame destí i una funció per pausar l'animació. Aquestes funcions venen representades per la següents definicions:

- **void PA_SetSpriteAnim(u8 screen, u8 sprite, s16 animframe);**
- **void PA_StartSpriteAnim(u8 screen, u8 sprite, s16 firstframe, s16 lastframe, s16 speed);**
- **PA_SpriteAnimPause(u8 screen, u8 sprite, u8 pause);**

  Screen: Atribut que informa de la pantalla on es mostrarà el text (1 per la pantalla superior i 0 per a la inferior).

  Sprite: Número de l'sprite a animar.

  Animframe: Número del frame dins de l'sprite.

  Firstframe, Lastframe: Número de frame inicial i final (respectivament) en l'animació.

  Pause: 1 pausat o 0 no pausat.

La primera funció defineix quin frame de l'sprite s'ha de mostrar en pantalla. Si mirem la Figura 17 de la pàgina 36, podem veure que per a la posició de frame 0 tenim un cofre tancat i l'hauríem de carregar amb aquest frame si volguéssim representar un element cofre tancat.

La segona funció defineix l'animació que ha de tenir un sprite. Les animacions són

cícliques i per tant es dóna un frame inicial i un frame final. L'animació passarà un a un per tots els frames que hi ha des de frame inicial fins a frame final i un cop arriba al frame final torna a iniciar des del frame inicial (veure Figura 18).

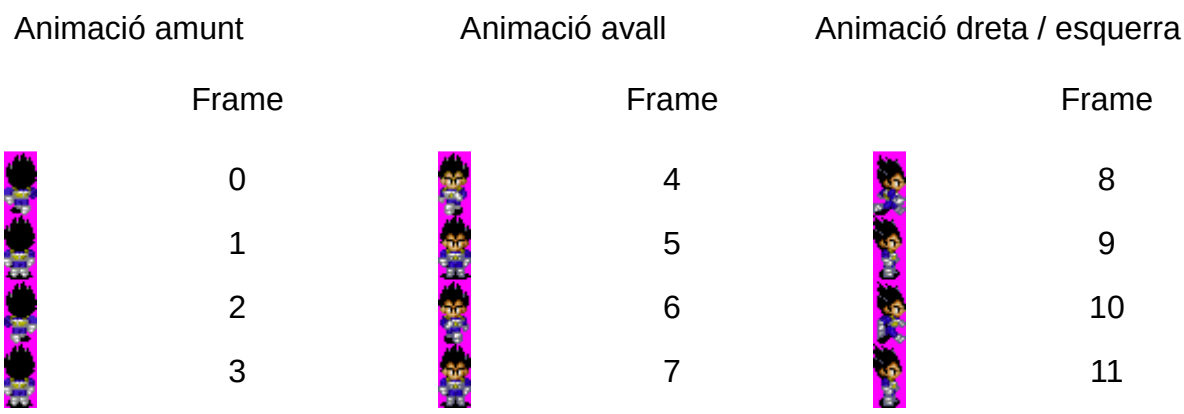


Figura 18: Esquema de frames d'un sprite.

Com podem veure a la Figura 18 anterior, les animacions estan definides de la següent manera:

- Frame 0 – 3: Animació de l'sprite caminant amunt.
- Frame 4 – 7: Animació de l'sprite caminant avall.
- Frame 8 – 11: Animació de l'sprite caminant dreta.

Per a realitzar les animacions cap a l'esquerra s'ha de fer un rotació de 180º respecte l'eix Y amb la següent funció:

- **PA_SetSpriteHflip(u8 screen, u8 sprite, u8 hflip);**

  h: Flip horitzontal, 1 activat 0 desactivat.

7.2.2.4.1.3 Desplaçament

Una de les parts més importants dels sprites és el desplaçament. Per a desplaçar els sprites es fa servir la següent funció:

- **PA_SetSpriteXY(u8 screen, u8 sprite, s16 x, s16 y);**

Screen: Atribut que informa de la pantalla on es mostrarà el text (1 per la pantalla superior i 0 per a la inferior).

Sprite: Número de l'sprite a animar.

X, Y: coordenades (respecte la pantalla) x i y en píxels, respectivament

7.2.2.4.2 Backgrounds

En aquest apartat treballarem amb els fons de tipus *Tiled* que són fons que tenen una estructura de tiles (cel·les de 8x8 píxels). Aquesta estructura serà molt útil per el tractament de col·lisions.

7.2.2.4.2.1 Inicialització i càrrega

La càrrega de fons és molt senzilla amb PAlib, cridant una única funció ja tenim carregat el fons en memòria:

- **PA_EasyBgLoad(screen, bg_number, bg_name);**

El que s'hauria de fer acte seguit de carregar el fons, és donar unes coordenades d'inici per a posicionar la pantalla correctament. Això es pot veure en el següent apartat.

7.2.2.4.2.2 Desplaçament

La funció principal dels fons en un motor de joc és l'scrolling (desplaçament del fons). És un procés igual que el desplaçament dels sprites. Aquesta funció ve donada per la següent definició de funció.

- **PA_EasyBgScrollXY(u8 screen, u8 bg_number, s16 x, s16 y);**

Screen: Atribut que informa de la pantalla on es mostrarà el text (1 per la pantalla superior i 0 per a la inferior).

Bg_number: Número del fons.

X, Y: coordenades (respecte el mapa) x i y en píxels, respectivament.

En la funció anterior es desplaça la pantalla pel mapa fins a la posició (x,y). A la Figura 19 podem veure un mapa amb les posicions del mapa (x,y) del mapa a la cantonada superior esquerra de la pantalla (requadre) i la coordenada (0,0) del mapa a la part superior esquerra del mapa.

Figura 19: Exemple de pantalla amb fons carregat i coordenades del mapa (X,Y).

7.2.2.4.2.1 Inicialització i càrrega

La càrrega de fons és molt senzilla amb PAlib, cridant una única funció ja tenim carregat el fons en memòria:

7.2.3 PAGfx

PAGfx és una aplicació Open Source que s'encarrega de convertir imatges en format mapa de bits a format de la Nintendo DS, per a que es pugui afegir directament al binari resultant (.nds). L'aplicació esta realitzada amb la llibreria.NET de Microsoft, per tant si volem executar-la en sistemes Linux s'haurà de fer a través del Projecte Mono.

Aquesta aplicació ja ve inclosa amb les utilitats adjuntes que porta incorporada les **PAlib**. Com podem veure a la Figura 20, l'aplicació de tractament d'imatges per a convertir en format propi de la Nintendo DS és molt intuïtiva. Es tria quin tipus de gràfic volem agregar (sprites o backgrounds) i els agreguem. També es tria el color de transparència (magenta per defecte). Un cop hem acabat s'ha de seleccionar *"Save and Convert"*.

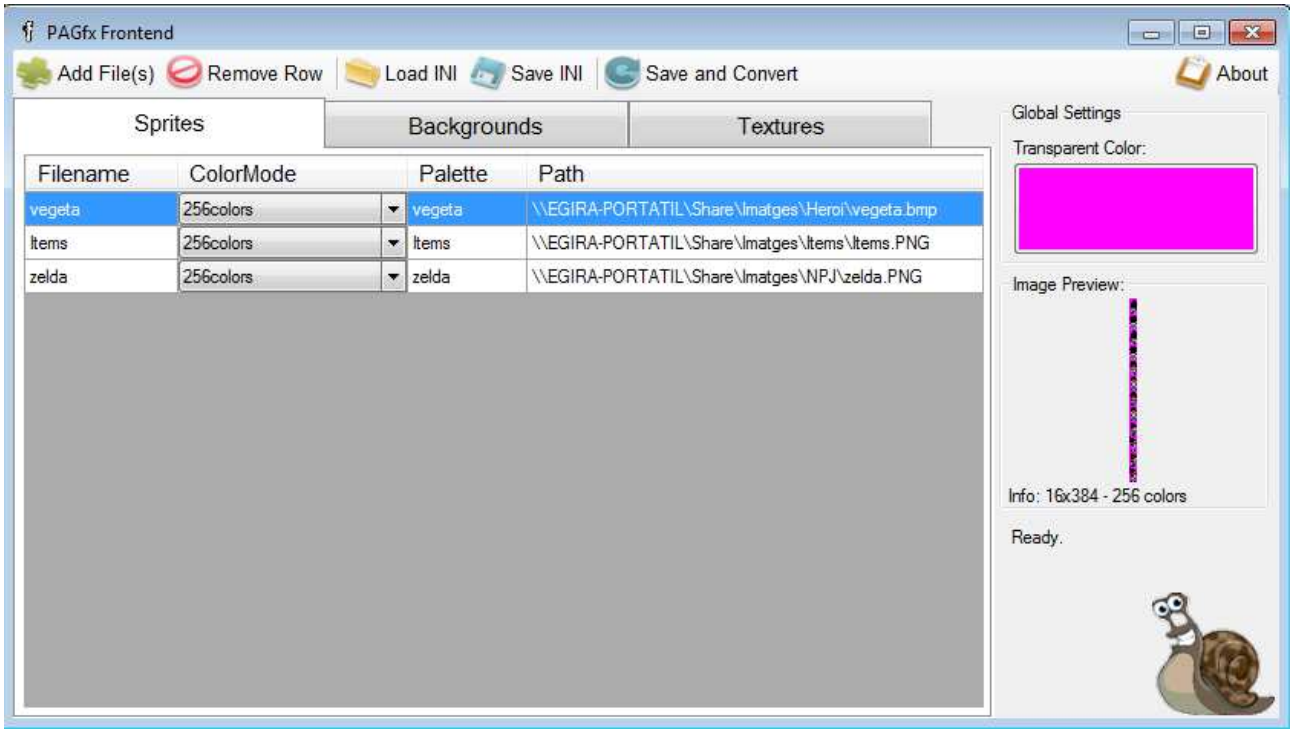


Figura 20: Interfície gràfica PAGfx.

PAGfx genera per a cada tipus:

- Sprite: Gráfics (*_Sprite.bin) i paleta (*_Pal.bin).
- Fons: Mapes (*_Map.bin), tiles (*_Tiles.bin), paletas (*_Pal.bin) i informació (*.c).
- Textures: textura (*_Texture.bin) y paleta (*_Pal.bin).

Per a tots ells també genera un arxiu all_gfx.h i un arxiu all_gfx.c. Tots aquests arxius s'han de guardar en un directori *gfx_all* dins la carpeta *source*, on hi ha tot el codi font (veure Figura 21).

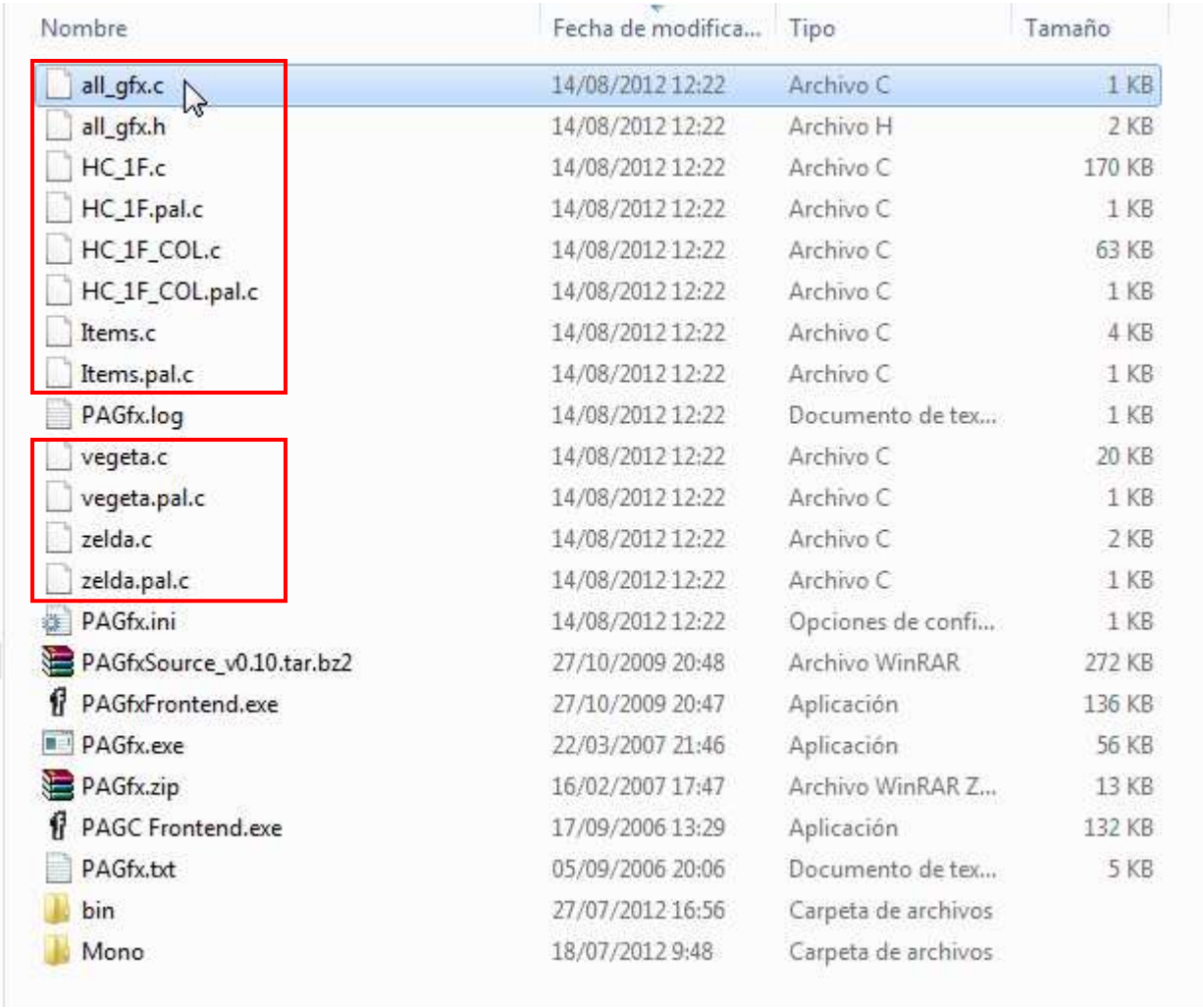


Figura 21: Arxius generats per PAGfx.

Per a cada recurs és important triar el tipus de conversió a realitzar:

- Per als sprites, la mida ha de ser múltiple de 8 (seguint les mides de la taula mostrada anteriorment) i amb un màxim de 256 colors.
- Per als fons, la seva dimensió horitzontal ha de ser múltiple de 256 píxels i la vertical de 192 píxels, que són les mides màximes de la pantalla de la Nintendo DS. Es triarà EasyBg per defecte i el programa triarà el mode més convenient per al projecte.
- Per a textures, els tipus només poden ser de 8 i 16 bits.

7.2.4 GIMP

GIMP és un programa de GNU per al tractament d'imatges (GNU Image Manipulation Program), creat per voluntaris, multiplataforma i distribuït sota la llicència GPL. GIMP serveix per processar gràfics i fotografies digitals. Els usos típics inclouen la creació de gràfics i logotips, el canvi de mida i retallat de fotografies, el canvi de colors, la combinació d'imatges usant capes, l'eliminació d'elements no desitjats de les imatges i la conversió entre diferents formats d'imatges. També es pot utilitzar el GIMP per crear imatges animades senzilles.

GIMP és conegut també per ser potser la primera gran aplicació lliure per a usuaris finals. Treballs anteriors, com Gcc, el nucli de Linux, etc. eren principalment eines de programadors per a programadors.

Aquest software s'ha utilitzat per a la creació del mapa de col·lisions i la edició dels diferents gràfics continguts en el projecte.

7.2.5 Notepad++

Notepad++ és un editor de codi font i el substitut de Notepad que suporta diversos llenguatges. S'executa en entorn de Microsoft Windows i el seu ús està regulat per la llicència GPL.

Basat en el potent component d'edició Scintilla, Notepad++ està escrit en C++ i utilitza l'API de Win32 i STL, el que assegura una velocitat major d'execució i mides més petites del programa. Mitjançant l'optimització de tantes rutines com sigui possible sense perdre la facilitat d'ús, Notepad++ intenta reduir les emissions mundials de

diòxid de carboni. Al utilitzar menys energia de la CPU, el PC pot moderar la marxa i reduir el consum d'energia, donant com a resultat un medi ambient més verd.

Aquest software s'ha utilitzat com a entorn de desenvolupament per la senzillesa, la presentació i ordenació del codi, i bàsicament perquè és un software molt lleuger.

7.2.6 LibreOffice

LibreOffice és un paquet d'ofimàtica Open Source desenvolupat per The Document Foundation sota la llicència GNU Lesser General Public License. Inclou eines com el processador de text, fulla de càlcul, presentacions, eines per el dibuix vectorial, base de dades i un editor de fórmules matemàtiques. Està disponible per GNU/Linux, Microsoft Windows i Mac OS X. És una suite que compleix amb l'estàndard ISO OpenDocument, per la qual cosa resultat senzill realitzar intercanvis de documents amb altres programes.

Aquest software s'ha fet servir per a redactar la memòria del projecte i dissenyar la presentació.

7.2.7 GanttProject

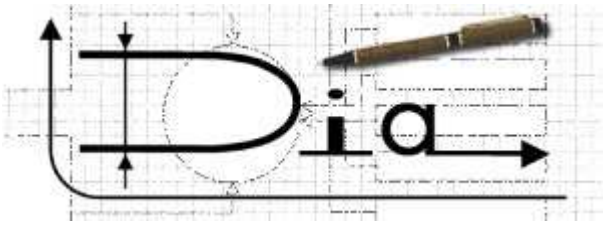
GanttProject és una aplicació de programari lliure que permet realitzar diagrames de Gantt. Un diagrama de Gantt és una eina gràfica, on l'objectiu és mostrar el temps de dedicació previst per a diferents tasques o activitats al llarg d'un temps total determinat.

El diagrama de Gantt no indica les relacions existents entre activitats, però la posició de cada tasca al llarg del temps fa que es puguin identificar aquestes relacions i

interdependències.

Bàsicament s'ha utilitzat aquest programari per a fer el gràfic de la temporalització del projecte.

7.2.8 Dia



Dia és un programa de codi lliure llicenciat sota llicencia GPL que forma part del projecte GNOME disponible tant per GNU/Linux com per Windows. És una aplicació de propòsit general per a la creació de diagrames de tot tipus, com per exemple diagrames d'entitat-relació, diagrames UML, diagrames de flux, diagrames de xarxes, de circuits elèctrics, etc. També permet exportar els diagrames en format SVG i PNG. Una de les utilitats que té aquest programari és la possibilitat de crear, a partir d'un fitxer UML, l'esquelet del codi font.

Aquest programari s'ha emprat per a fer tots els diagrames d'UML presents en aquesta documentació.

8. Anàlisi i disseny del sistema

En aquest apartat es descriurà en què consisteix aquest projecte abans d'entrar en detalls tècnics. Amb aquesta explicació, es pretén que el capítol de Disseny i Implementació sigui més entenedor i fàcil de seguir.

8.1 Descripció general

El motor de joc que volem desenvolupar en aquest projecte és un motor de joc pensat per als jocs RPG a l'estil Pokémon o Zelda (veure Figura 22 i 23). Aquest estil de joc es basa en el control d'un personatge en tercera persona on haurem de seguir una història amb un argument molt desenvolupat i amb molts escenaris, NPC's i objectes amb els quals podrem interactuar. Cal remarcar que un dels principals atractius és la no-linealitat del joc, és a dir, hi haurà moments en el joc que podrem decidir quines missions volem fer i quines no.

Figura 22: Imatge del joc Pokémon.

Figura 23: Imatge del joc Zelda: A link to the past.

8.2 Identificació dels actors

A continuació s'identifiquen els actors de l'aplicació. Un actor és una entitat externa (persona, sistema, subsistema, etc) que interactua amb el sistema interpretant un determinat rol o estat.

En la part del videojoc (usuari final), no hi ha cap tipus de manteniment, al tractar-se d'un programa en el qual no existeix distinció entre els possibles usuaris que el poden utilitzar. D'altra banda, existeix l'usuari amb un rol de programador que farà servir la llibreria que s'està desenvolupant per a poder fer videojocs de manera senzilla, però la documentació s'ha orientat a l'usuari final (veure Figura 24).

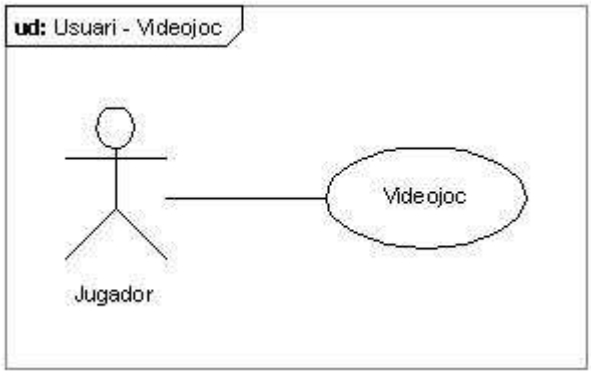


Figura 24: Identificació dels actors.

Com es pot veure en la figura anterior, l'esquema resultant és senzill ja que tots els usuaris que utilitzin l'aplicació disposaran dels mateixos privilegis.

8.3 Casos d'ús

En enginyeria del programari, un cas d'ús és una tècnica per a la captura de requisits potencials d'un nou sistema o una actualització de programari. Cada cas d'ús proporciona un o més escenaris que indiquen com hauria d'interactuar el sistema amb l'usuari o amb un altre sistema per aconseguir un objectiu específic. Normalment, en els casos d'ús s'evita l'ús d'argots tècnics, preferint en el seu lloc un llenguatge més proper a l'usuari final.

Al iniciar el videojoc, trobem un menú que conté 2 opcions i l'usuari ha de triar una de les dues opcions: *Iniciar Partida* i *Sobre el programari* (veure Figura 25).

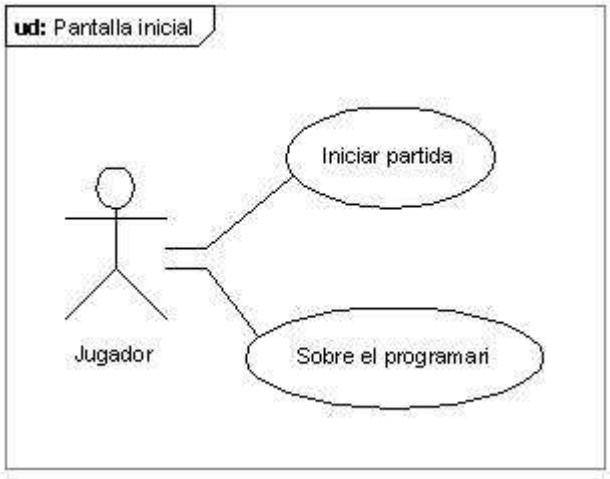


Figura 25: Diagrama de cas d'ús: Pantalla inicial.

Fitxa de cas d'ús Iniciar partida	
Descripció	El jugador inicia una partida.
Actor	Jugador
Precondició	Cert
Flux Principal	1- Prémer Start.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	S'ha arrancat el motor de joc.

Fitxa cas d'ús Sobre el programari	
Descripció	Veure informació referent al programari.
Actor	Jugador
Precondició	Cert
Flux Principal	1- Prémer Select.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	Es mostra en pantalla informació referent al programari.

A la Figura 26, es pot veure el cas d'ús general que es produeix al iniciar una partida de joc.

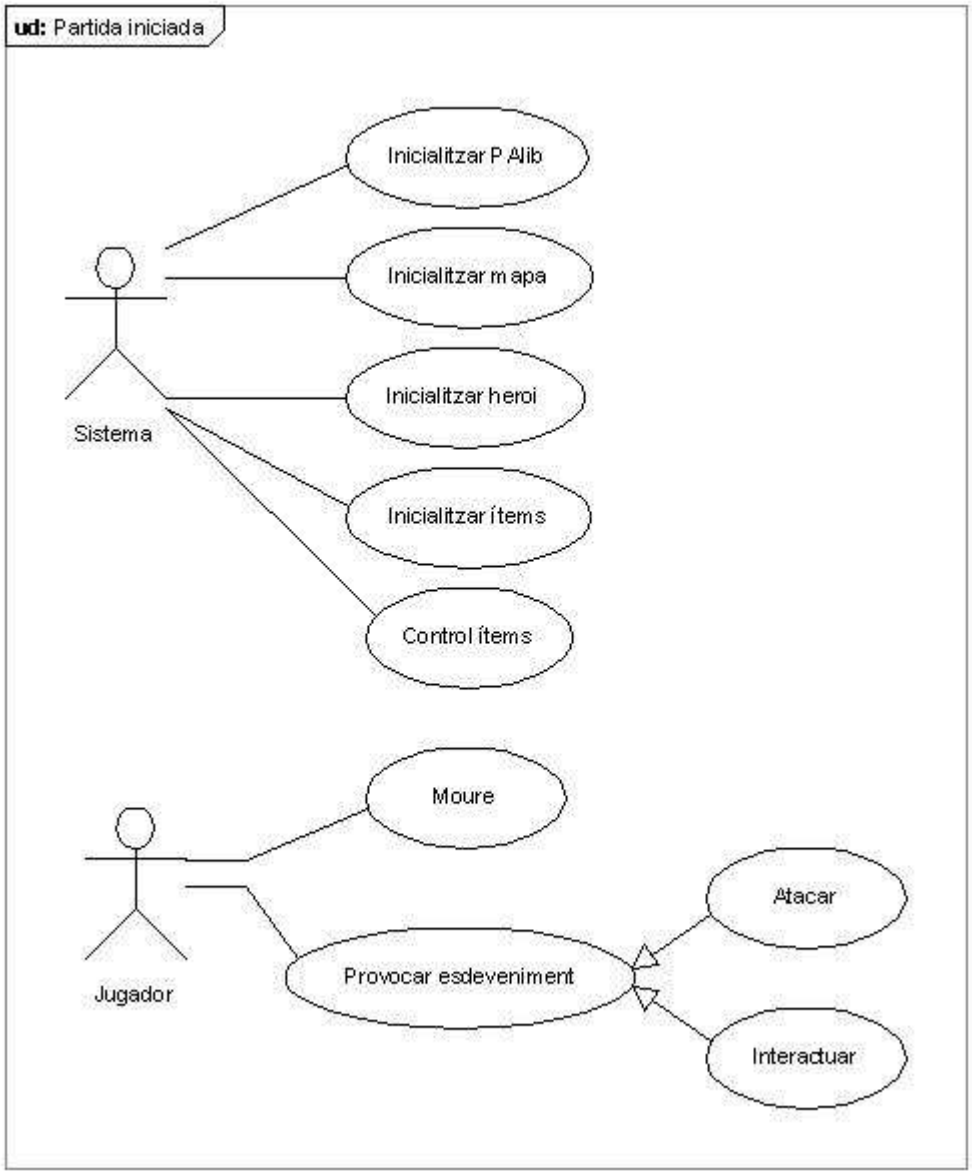


Figura 26: Diagrama de cas d'ús: Partida iniciada.

Les següents fitxes de cas d'ús es dediquen bàsicament a la càrrega del motor de joc. El sistema s'encarrega de carregar totes les llibreries necessàries per a la càrrega i tractament de gràfics, eines per a mostrar text i inicialitzadors propis dels elements de l'escena.

Fitxa cas d'ús Inicialitzar PAlib	
Descripció	S'inicia la llibreria externa **PAlib**.
Actor	Sistema
Precondició	Partida inicialitzada.
Flux Principal	1- El sistema carrega en memòria la llibreria **PAlib**.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	S'ha inicialitzat la llibreria externa **PAlib**.

Fitxa cas d'ús Inicialitzar mapa	
Descripció	Es carreguen en memòria tots els atributs del mapa: mides, posició inicial i imatge.
Actor	Sistema
Precondició	Partida inicialitzada.
Flux Principal	1- Es crea un objecte Mapa amb una sèrie de paràmetres definits. 2- Es carrega en memòria la imatge mitjançant la llibreria **PAlib**.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	S'han carregat en memòria tots els atributs referents al mapa.

Fitxa cas d'ús Inicialitzar heroi	
Descripció	Es carreguen en memòria tots els atributs de l'heroi: posició inicial, imatge, atributs d'animació.
Actor	Sistema
Precondició	Partida inicialitzada.
Flux Principal	1- Es crea un objecte Heroi amb una sèrie de paràmetres definits. 2- Es carrega en memòria la imatge mitjançant la llibreria **PAlib**.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	S'han carregat en memòria tots els atributs referents a l'heroi.

Fitxa cas d'ús Inicialitzar ítems	
Descripció	Es creen tots els elements de l'escena amb els respectius atributs i es carreguen en un vector d'elements.
Actor	Sistema
Precondició	Partida inicialitzada.
Flux Principal	1- Es creen els elements amb els paràmetres predefinits . 2- Es carrega en memòria la imatge mitjançant la llibreria **PAlib**. 3- S'afegeix l'element al vector d'elements que conté l'objecte mapa.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	S'han creat i afegit tots els elements que formen part de l'escena.

Un cop inicialitzada la llibreria **PAlib**, el mapa, l'heroi i els elements de l'escena, l'usuari ja pot començar a interactuar amb el sistema. A l'interactuar amb el sistema es produeixen els següents casos d'ús.

Fitxa cas d'ús Moure	
Descripció	Desplaçar l'heroi per el mapa
Actor	Jugador.
Precondició	Llibreria PAlib inicialitzada, mapa inicialitzat, heroi inicialitzat.
Flux Principal	1- Es mou el personatge principal mitjançant el pad direccional.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	El jugador s'ha desplaçat pel mapa.

Fitxa cas d'ús Provocar esdeveniment	
Descripció	L'heroi provoca un esdeveniment a un element de l'escena.
Actor	Jugador
Precondició	L'element està en pantalla. L'heroi està a una distància d (predeterminada) o menys de l'element a interactuar.
Flux Principal	1- L'heroi provoca un canvi d'estatus a l'element via un esdeveniment programat. 2- L'element executa la seva acció predefinida.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	L'element ha executat la seva acció predefinida.

Fitxa cas d'ús Control Ítems	
Descripció	Avalua respecte la posició de l'heroi si els elements del mapa han d'intervenir en l'escena i si algun esdeveniment ha provocat l'execució de les seves respectives accions.
Actor	Sistema
Precondició	Llibreria PAlib inicialitzada, mapa inicialitzat, heroi inicialitzat.
Flux Principal	1- Per cada element del mapa avaluar si ha d'estar en pantalla. 1.1- Si no ha d'estar en pantalla. 1.1.1- Esborrar element de l'escena. 1.2- Si ha d'estar en pantalla i no està mostrat. 1.2.1- Mostrar element a l'escena. 1.3- Si ha d'estar en pantalla i està mostrat. 1.3.1- Avaluar si algun esdeveniment actua sobre aquest element i executar la seva acció associada.
Subfluxos	Cap.
Fluxos alternatius	Cap.
Postcondició	S'ha recorregut el vector d'elements del mapa i s'han tractat segons l'esdeveniment programat.

8.4 Diagrames d'activitat

El diagrama d'activitat mostra el desenvolupament de l'execució del programa. S'ha dividit el diagrama en dos parts per a que sigui més comprensible (Figures 27 i 28).

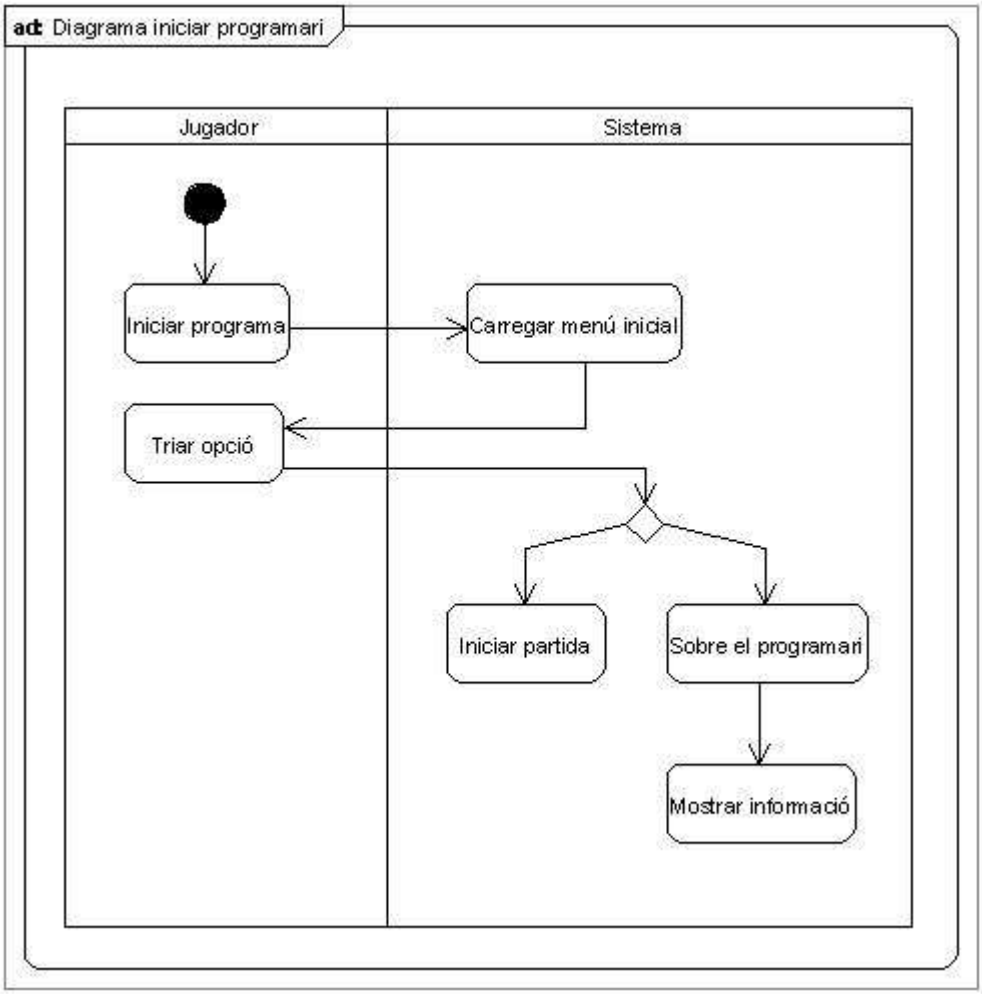


Figura 27: Diagrama d'activitat d'inici del programa.

A la figura anterior es pot veure el diagrama d'activitat del menú inicial. Quan l'usuari inicia el programari, el sistema carrega un menú principal on l'usuari haurà de triar unes de les opcions mostrades. Si es tria la opció *Sobre el programari* el sistema mostrarà informació sobre el programari. Si es tria la opció *Iniciar partida*, el sistema iniciarà el videojoc seguint el flux mostrat a la Figura 28.

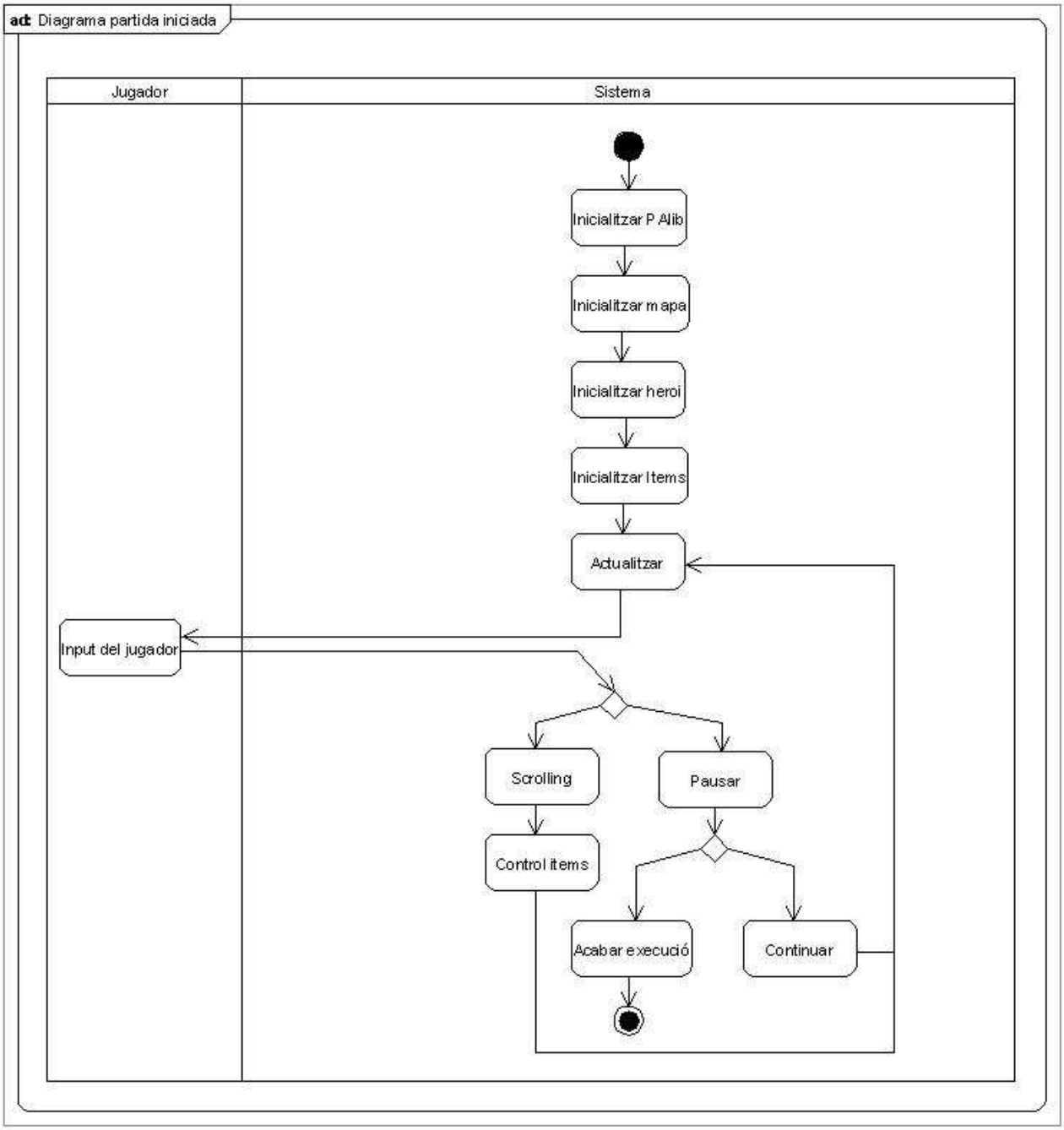


Figura 28: Diagrama d'activitat d'una partida iniciada.

A la figura anterior podem veure l'execució que segueix *Iniciar Partida*. Per a cada input del jugador s'avalua quina tecla a polsat. Si la tecla polsada és *Start*, es pausa l'execució i es mostra un menú on l'usuari pot triar si continuar la partida o la vol finalitzar. Si per contra la tecla polsada és una altra, s'avalua quina tecla s'ha polsat i quin esdeveniment s'ha de tractar, per exemple moure l'heroi o provocar una interacció amb un element de l'escenari.

S'ha d'aclarir que l'acció *Actualitzar* és un bucle on l'aplicació no es dóna per aturada

a no ser que es faci des del menú associat a la tecla *Start*.

8.5 Diagrama de classes

Un diagrama de classes és un tipus de diagrama estàtic que descriu l'estructura d'un sistema mostrant les seves classes, atributs i relacions entre ells. Els diagrames de classes són utilitzats durant el procés d'anàlisi i disseny dels sistemes, on es crea el disseny conceptual de la informació que tindrà el sistema, i els components que s'encarreguen del funcionament i de la relació entre un i l'altre.

El diagrama de classes de la Figura 29 mostra el conjunt de classes que formen el motor de joc. S'ha implementat un disseny de capes orientat a objectes amb herència i polimorfisme un disseny per a oferir un motor de joc modulable i fàcil de modificar. D'aquesta manera cada classe tindrà la seva pròpia lògica de funcionament.

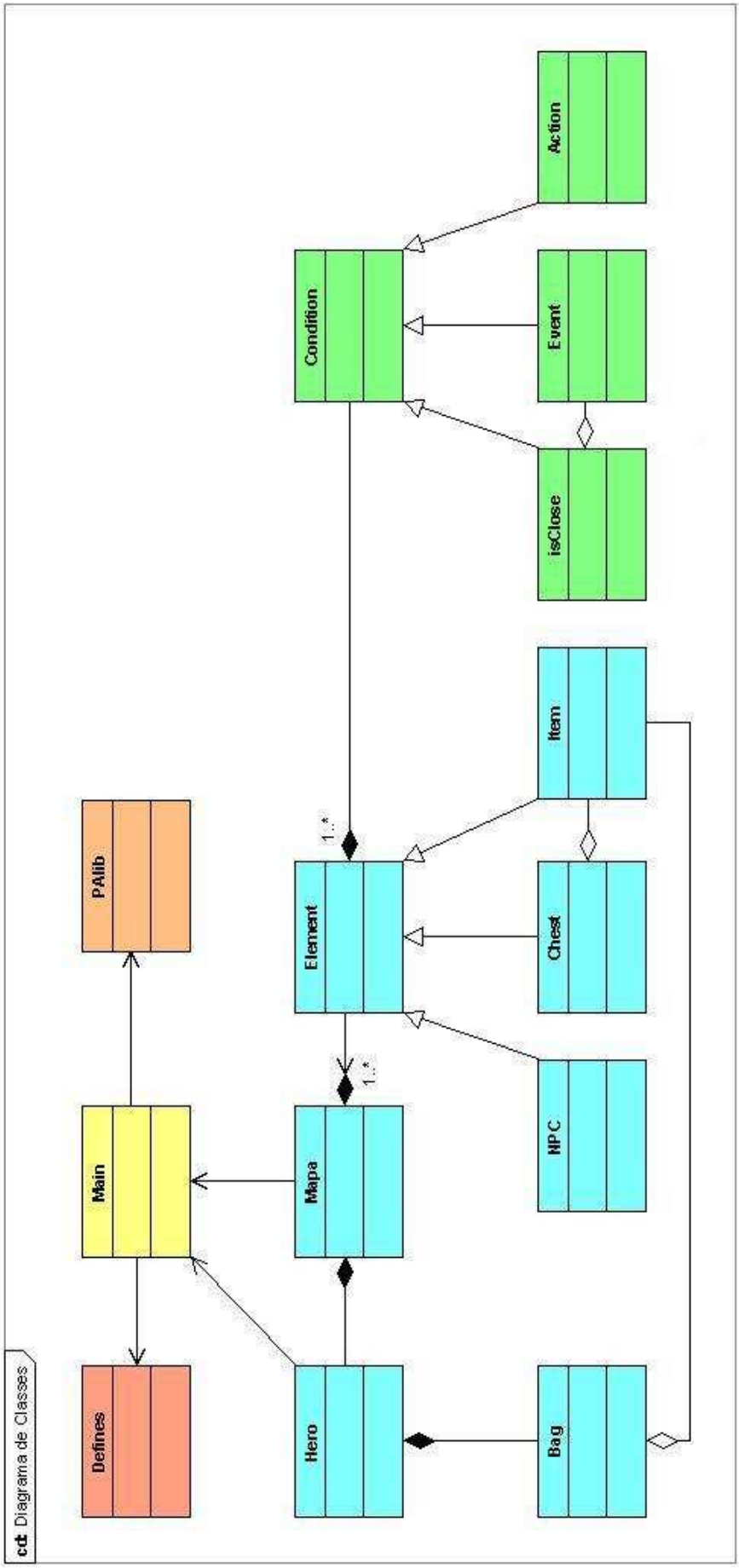


Figura 29: Diagrama de classes del motor de joc.

Com podem veure en la figura anterior, el diagrama s'ha estructurat en 5 parts o colors:

- Color groc: Classe interfície, on l'usuari interactua amb tot el motor de joc mitjançant les entrades en el pad de la consola.
- Color vermell: Paràmetres configurables del motor de joc.
- Color taronja: Llibreria externa PAlib.
- Color blau: Estructura de dades mínima per a poder desenvolupar un joc de tipus RPG.
- Color verd: Motor d'esdeveniments, on es defineix quan i què ha de fer cada element de l'escena.

En el següent apartat s'explicaran les classes amb més detall.

8.5.1 Classe Main

Aquesta és la classe principal del projecte. És l'encarregada d'inicialitzar la llibreria **PAlib**, carregar el motor de joc i interactuar amb l'usuari capturant els esdeveniments provocats per la utilització del pad de la consola. Aquesta classe fa servir els paràmetres definits a la classe *Defines*.

Main
+ numSprites: s32 + numFondo: s32 + numPaleta: u16 + Heroi: Hero + Map: Mapa
+ pantallaInicial(); + initBackground(Mapa m) + initItem(Mapa m, s32 numSprite, s32 numPaleta) + initHeroi(Hero p) + animacioBasica(Hero p) + scrolling(Mapa m, Hero p)

+ scrollingMapaHoritzontal(Mapa m, Hero p)
+ scrollingMapaVertical(Mapa m, Hero p)
+ scrollingPersonatgeHoritzontal(Mapa m, Hero p)
+ scrollingPersonatgeVertical(Mapa m, Hero p)
+ llistarInventari()
+ opcionsItems()
+ bool get1Tecla()
+ u16 getKeyPressed()
+ s32 getTileXPlayer(Mapa m, Hero p)
+ s32 getTileYPlayer(Mapa m, Hero p)
+ s32 getCollision(s32 tileX, s32 tileY)
+ bool canMoveLeft(s32 tileX, s32 tileY)
+ bool canMoveRight(s32 tileX, s32 tileY)
+ bool canMoveUp(s32 tileX, s32 tileY)
+ bool canMoveDown(s32 tileX, s32 tileY, Hero p)
+ ctrlItems(Mapa m, Hero p)

Atributs:

+ numSprites: Variable de tipus enter de 32 bits, defineix el número d'sprites en pantalla.

+ numFondo: Variable de tipus enter de 32 bits, defineix el número de fondos del motor de joc.

+ numPaleta: Variable de tipus enter de 16 bits, defineix el número de paletes.

+ Heroi: Variable de tipus Heroi amb paràmetres carregats de la classe Defines.

+ Map: Variable de tipus Mapa amb paràmetres carregats de la classe Defines. Contindrà tots els elements que intervindran en l'escena.

Mètodes:

+ void pantallaInicial(): Mostra el menú inicial al arrancar el programari.

+ initBackground(Mapa m): Inicialitzador de l'objecte de la classe Mapa.

+ initHeroi(Hero p): Inicialitzador de l'objecte de la classe Heroi.

+ initItem(Mapa m, s32 numSprite, s32 numPaleta): Inicialitzador dels elements Item, s'agreguen a un vector de l'objecte mapa.

+ animacioBasica(Hero p): Mètode per a fer l'animació gràfica de l'objecte heroi.

El pseudocodi d'aquest mètode és el següent:

```
SI PadAmuntPolsat LLAVORS
  SI framePersonatge >= 3 LLAVORS
    framePersonatge = 0
  FSI
  ferAnimacioPersonatge(PantallaAbaix, Heroi, 0, 3)
FSI

SI PadAvallPolsat LLAVORS
  SI framePersonatge >= 3 LLAVORS
    framePersonatge = 0
  FSI
  ferAnimacioPersonatge(PantallaAbaix, Heroi, 0, 3)
FSI

SI PadDretaPolsat LLAVORS
  SI framePersonatge >=11 LLAVORS
    framePersonatge = 8
  FSI
  ferAnimacioPersonatge(PantallaAbaix, Heroi, 8, 11)
FSI

SI PadEsquerraPolsat LLAVORS
  SI framePersonatge >=11 LLAVORS
    framePersonatge = 8
  FSI
  ferAnimacioPersonatge(PantallaAbaix, Heroi, 8, 11)
  HflipSprite(PantallaAbaix, Heroi)
FSI

SI PadAmuntPolsat O PadAvallPolsat O PadDretaPolsat O
PadEsquerraPolsat LLAVORS
  PausarAnimacio()
FSI
```

El que fa aquest mètode és avaluar quina tecla s'està polsant i en base a això s'executa una seqüència d'animació entre els Frames que es corresponguin amb els gràfics de la Figura 18.

+ scrolling(Mapa m, Hero p): Mètode per al moviment de l'heroi o el mapa.

El pseudocodi d'aquest mètode és el següent:

```
SI posicioMapaX < (tamanyMapaX DIV 2) LLAVORS
  SI posicioMapaX == 0 LLAVORS
    scrollingPersonatgeHoritzontal(mapa, personatge)
    SI posicioPersonatgeX == migPantallaX LLAVORS
      incrementoPosicioMapaX
    FSI
  ALTRAMENT
    scrollingMapaHoritzaontal(mapa, personatge)
  FSI
ALTRAMENT SI posicioMapaX >= (tamanyMapaX DIV 2)
LLAVORS
  SI posicioMapaX == tamanyMapaX LLAVORS
    scrollingPersonatgeHoritzontal(mapa, personatge)
    SI posicioPersonatgeX == migPantallaX LLAVORS
      decrementoPosicioMapaX
    FSI
  ALTRAMENT
    scrollingMapaHoritzontal(mapa, personatge)
  FSI

SI posicioMapaY < (tamanyMapaY DIV 2) LLAVORS
  SI posicioMapaY == 0 LLAVORS
    scrollingPersonatgeVertical(mapa, personatge)
    SI posicioPersonatgeY == migPantallaY LLAVORS
      incrementoPosicioMapaY
    FSI
  ALTRAMENT
    scrollingMapaVertical(mapa, personatge)
  FSI
ALTRAMENT SI posicioMapaY >= (tamanyMapaY DIV 2)
LLAVORS
  SI posicioMapaY == tamanyMapaY LLAVORS
    scrollingPersonatgeVertical(mapa, personatge)
    SI posicioPersonatgeY == migPantallaY LLAVORS
      decrementoPosicioMapaY
    FSI
  ALTRAMENT
    scrollingMapaVertical(mapa, personatge)
  FSI
```

Bàsicament, el que fa aquest codi és avaluar a quina meitat de la pantalla es troba l'heroi i avalua si qui s'ha de moure és el fons o l'heroi. A l'hora de fer l'scrolling trobem dos situacions: la primera, quan qui es mou és el fons i l'heroi queda centrat en el mapa. La segona situació és quan el mapa arriba a un dels extrems, llavors el mapa queda fix i qui es mou és l'heroi.

A la Figura 30 podem veure com l'heroi està centrat a la pantalla i qui es mou és el fons.

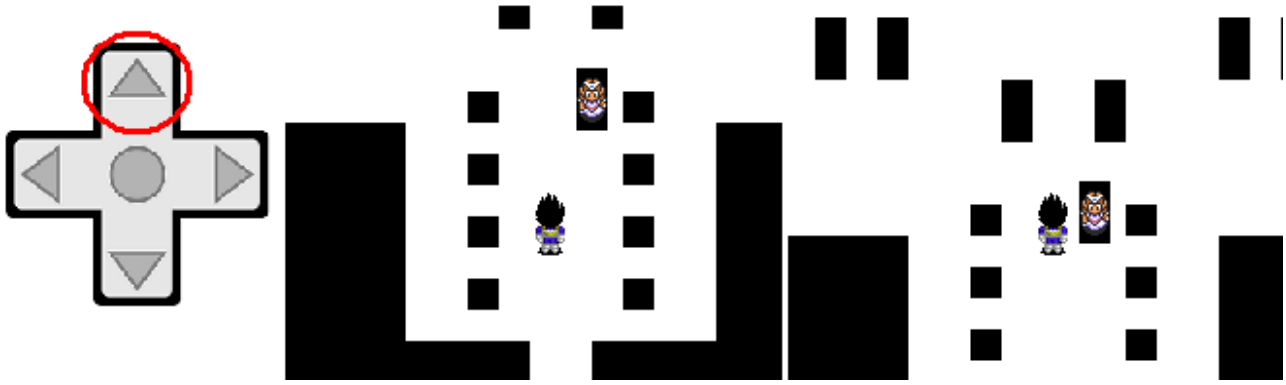


Figura 30: Exemple d'scrolling amb el moviment del fons.

A la Figura 31 podem veure com l'heroi es desplaça del centre de la pantalla i el fons queda fix.

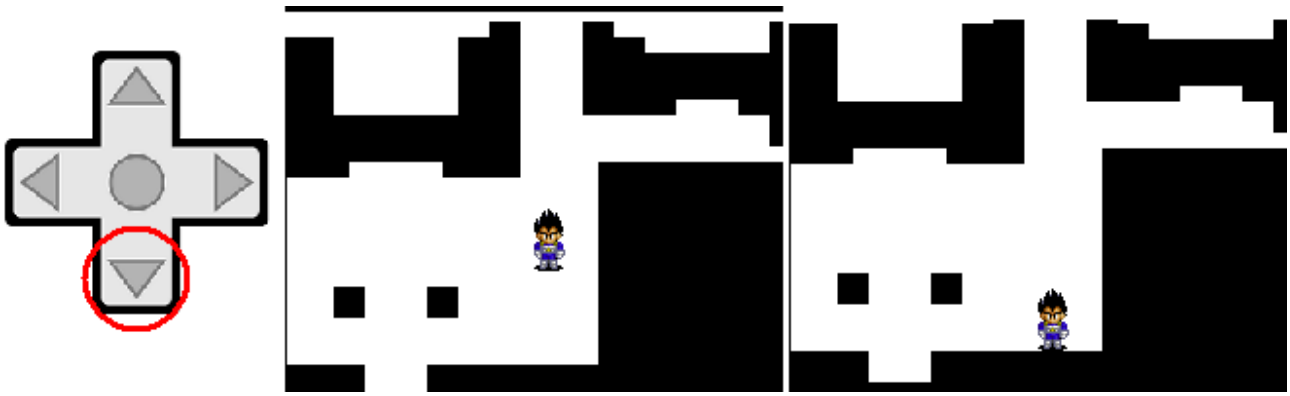


Figura 31: Exemple d'scrolling amb el moviment de l'heroi.

+ scrollingMapaHoritzontal(Mapa m, Hero p): Mètode per a moure el mapa en sentit horitzontal.

El pseudocodi d'aquest mètode és el següent:

```
SI posicioMapaX + PadDretaPolsat – PadEsquerraPolsat <= 0 I
posicioMapaX + PadDretaPolsat – PadEsquerraPolsat <=
tamanyMapaX LLAVORS
  SI PadDretaPolsat LLAVORS
    SI pucMoureALaDreta LLAVORS
      moureMapaX(posicioMapaX + PadDretaPolsat –
PadEsquerraPolsat)
    FSI
  FSI

  SI PadEsquerraPolsat LLAVORS
    SI pucMoureALaEsquerra LLAVORS
      moureMapaX(posicioMapaX + PadDretaPolsat –
PadEsquerraPolsat)
    FSI
  FSI

  ActualitzarPosicioMapaX(PantallaAbaix, fondoMapa,
posicioMapaX, posicioMapaY)
FSI
```

El que fa aquest codi és primer avaluar si la següent posició del mapa és una posició que està dins els límits del mapa. Això és calcula amb la següent fòrmula:

posicioMapaX + *PadDretaPolsat* – *PadEsquerraPolsat*

- PosicioMapaX: Posició X del mapa.
- PadDretaPolsat: Aquesta variable retorna 1 o 0 segons estigui polsat o no el pad direccional dret.
- PadEsquerraPolsat: Aquesta variable retorna 1 o 0 segons estigui polsat o no el pad direccional esquerra.

Si volem desplaçar-nos a la dreta s'incrementarà la posicióMapaX amb la variable PadDretaPolsat (que tindrà valor 1) i es decrementarà amb la variable PadEsquerraPolsat (que tindrà valor 0); si volem desplaçar-nos a l'esquerra el funcionament serà exactament el mateix però amb els valors de las variables PadDretaPolsat = 0 i PadEsquerraPolsat = 1.

El següent que es fa és avaluar si el pad esquerra / dreta està polsat i en cas afirmatiu, s'avalua si es pot moure a la següent tile mitjançant la funció *pucMoureALaDreta* / *pucMoureALaEsquerra*. Aquesta funció el que fa és avaluar

si la següent tile és una tile lliure, és a dir, no hi detecta cap col·lisió. En el cas que no detecti cap col·lisió, s'actualitza la posició X del mapa i es fa el desplaçament corresponent del mapa.

En la següent figura es pot veure gràficament el diagrama d'activitat associat al pseudocodi.

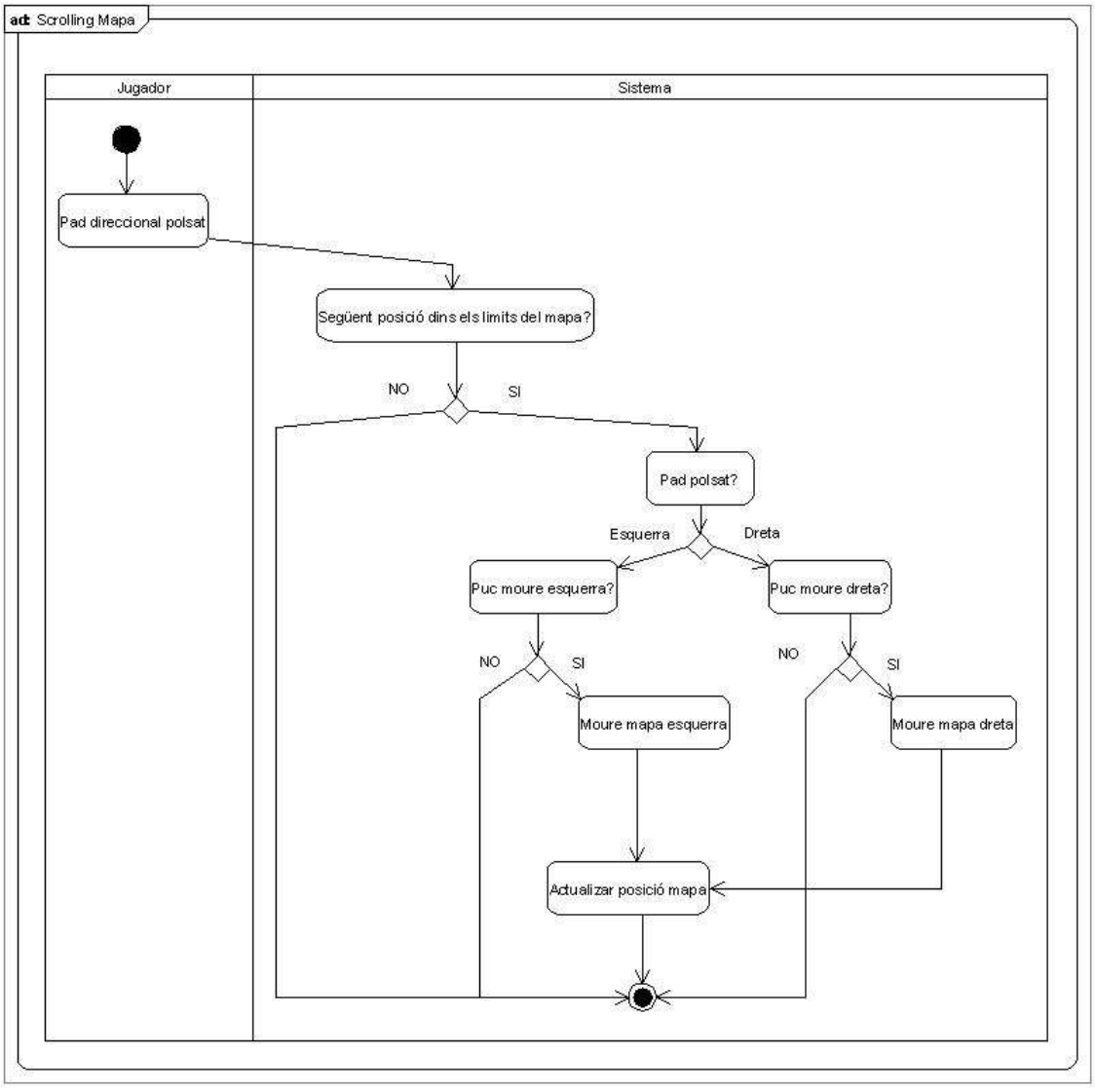


Figura 32: Diagrama d'activitat de l'scrolling mapa horitzontal.

+ scrollingMapaVertical(Mapa m, Hero p): Mètode per a moure el mapa en sentit vertical. El funcionament és igual que el mètode anterior *scrollingMapaHoritzontal(Mapa m, Hero p).*

+ scrollingPersonatgeHoritzontal(Mapa m, Hero p): Mètode per a moure l'heroi

per la pantalla en sentit horitzontal.

El pseudocodi d'aquest mètode és el següent:

```
movY = posicióPersonatgeY
movX = posicióPersonatgeX + PadDretaPolsat –
PadEsquerraPolsat

SI movX >=0 I movX < 240 LLAVORS
  SI PadDretaPolsat LLAVORS
    SI pucMoureALaDreta LLAVORS
      actualitzarPosicioPersonatgeX(movX)
    FSI
  FSI

  SI PadEsquerraPolsat LLAVORS
    SI pucMoureALaEsquerra LLAVORS
      actualitzarPosicioPersonatgeX(movX)
    FSI
  FSI
  moureSpriteXY(PantallaAbaix,nSprite, movX, movY)
FSI
```

El que fa aquest codi és primer avaluar si la següent posició del personatge és una posició que està dins d'els límits de la pantalla. Això és calcula amb la següent fòrmula:

posicioPersonatgeX + *PadDretaPolsat* – *PadEsquerraPolsat*

- PosicioMapaX: Posició X del personatge a la pantalla.
- PadDretaPolsat: Aquesta variable retorna 1 o 0 segons estigui polsat o no el pad direccional dret.
- PadEsquerraPolsat: Aquesta variable retorna 1 o 0 segons estigui polsat o no el pad direccional esquerra.

Si volem desplaçar-nos a la dreta s'incrementarà la posicióPersonatgeX amb la variable PadDretaPolsat (que tindrà valor 1) i es decrementarà amb la variable PadEsquerraPolsat (que tindrà valor 0); si volem desplaçar-nos a l'esquerra el funcionament serà exactament el mateix però amb els valors de las variables PadDretaPolsat = 0 i PadEsquerraPolsat = 1.

El següent que es fa és avaluar si el pad esquerra / dreta està polsat i en cas afirmatiu s'avalua si es pot moure a la següent tile mitjançant la funció *pucMoureALaDreta / pucMoureALaEsquerra*. Aquesta funció el que fa és avaluar si la següent tile és una tile lliure, és a dir, no hi detecta cap col·lisió. En el cas que no detecti cap col·lisió, s'actualitza la posició X del personatge i es fa el desplaçament corresponent del personatge.

En la següent figura es pot veure gràficament el diagrama d'activitat associat al pseudocodi.

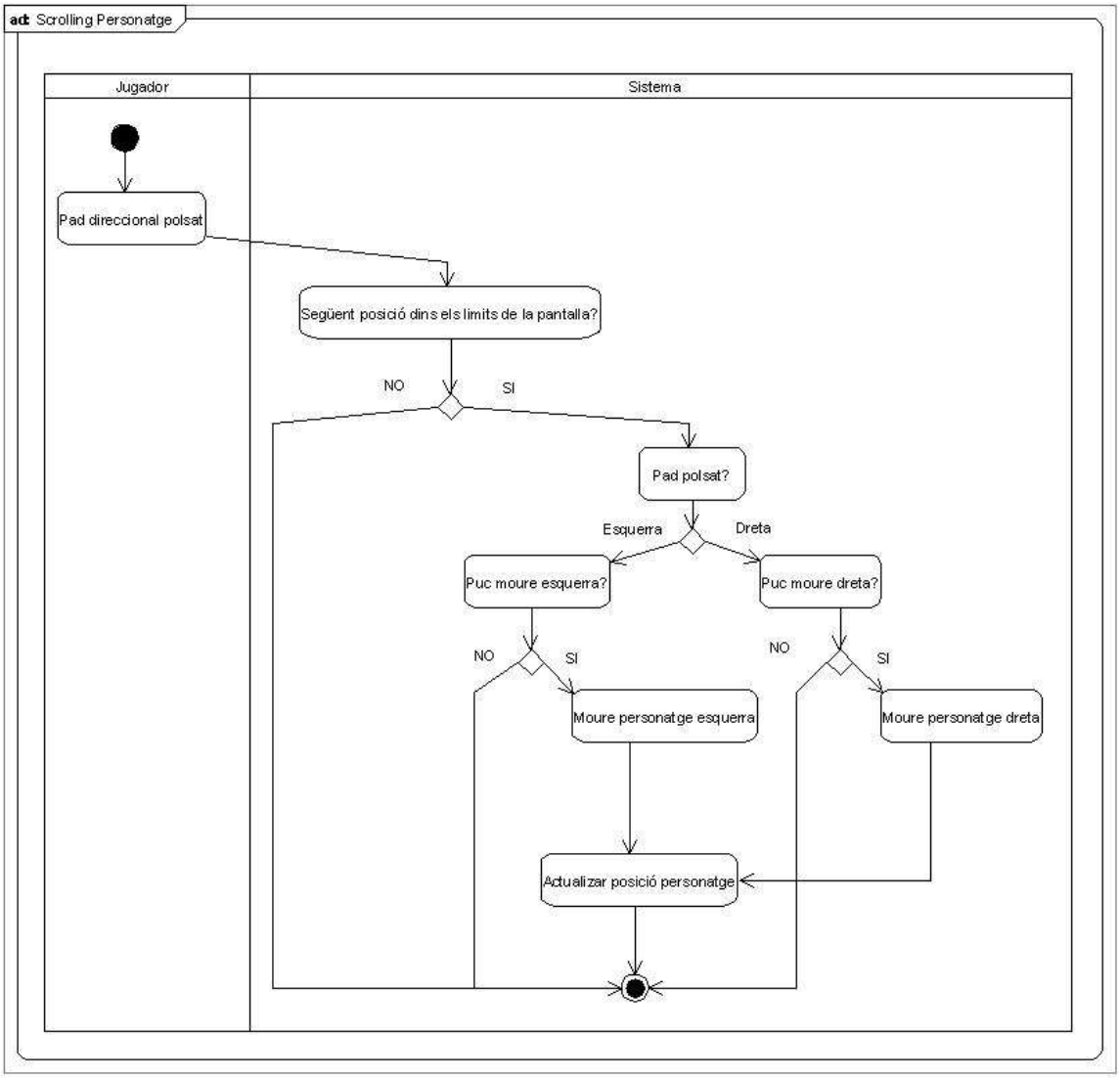


Figura 33: Diagrama d'activitat de l'scrolling personatge horitzontal.

+ scrollingPersonatgeVertical(Mapa m, Hero p): Mètode per a moure l'heroi per la pantalla en sentit vertical. El funcionament és igual que el mètode anterior

scrollingPersonatgeHoritzontal(Mapa m, Hero p).

+ llistarInventari(): Mètode que mostra per pantalla els ítems que conté la motxilla del personatge principal.

+ opcionsItems(): Mètode que mostra per pantalla les opcions que permet triar l'ítem amb el qual interactuem.

+ bool get1Tecla(): Mètode que indica si s'està polsant només una tecla o varies tecles a la vegada.

+ u16 getKeyPressed(): Mètode que retorna quina tecla s'està polsant.

+ s32 getTileXPlayer(Mapa m, Hero p): Mètode que indica quina tile de la coordenada X ocupa l'heroi.

+ s32 getTileYPlayer(Mapa m, Hero p): Mètode que indica quina tile de la coordenada Y ocupa l'heroi.

+ s32 getCollision(s32 tileX, s32 tileY): Mètode que retorna el valor de la col·lisió per a unes coordenades (en tiles).

+ bool canMoveLeft(s32 tileX, s32 tileY): Mètode que diu si respecte unes coordenades (en tiles), podem desplaçar el personatge a l'esquerra fins a la següent tile.

+ bool canMoveRight(s32 tileX, s32 tileY): Mètode que diu si respecte unes coordenades (en tiles), podem desplaçar el personatge a la dreta fins a la següent tile.

+ bool canMoveUp(s32 tileX, s32 tileY): Mètode que diu si respecte unes coordenades (en tiles), el podem desplaçar a dalt fins a la següent tile.

+ bool canMoveDown(s32 tileX, s32 tileY, Hero p): Mètode que diu si respecte unes coordenades (en tiles), el podem desplaçar avall fins a la següent tile.

+ ctrlItems(Mapa m, Hero p): Mètode que controla quins elements han d'estar visualitzats i quins no. També controla sobre quins elements de l'escena s'hi provoquen certs esdeveniments.

El pseudocodi d'aquest mètode és el següent:

```
PER (cada element del mapa) FER
  SI Element_no_ha_d'estar_en_pantalla LLAVORS
    Borrar_grafic_pantalla
```

```
ALTRAMENT SI Element_ ha_d'estar_en_pantalla LLAVORS
  SI Element_en_pantalla_i_no_visible LLAVORS
    Mostro_Element
  ALTRAMENT SI Element_en_pantalla_i_visible LLAVORS
    SI genero_event LLAVORS
      SI es_un_cofre LLAVORS
        SI està_tancat LLAVORS
          Fer_acció_cofre
        FSI
      ALTRAMENT SI es_un_item LLAVORS
        SI tinc_espai_a_la_motxilla LLAVORS
          Mostro_opcions
          SI opcio_triada_utilitzar LLAVORS
            Fer_accio_item
          ALTRAMENT SI opcio_triada_guardar LLAVORS
            Guardar_item_a_bag
          FSI
        ALTRAMENT
          Fer_accio_item
        FSI
      ALTRAMENT SI es_un_npc LLAVORS
        SI no_es_dinamic LLAVORS
          Fer_accio_npc
        ALTRAMENT
          Fer_accio_enemic
          SI enemic_mort LLAVORS
            Borrar_grafic_pantalla
            Borrar_element_vectorElements
          ALTRAMENT SI heroi_mort LLAVORS
            Mostrar_missatge_fi
          FSI
          Actualitzar_element_vectorElements
        FSI
      FSI
    FSI
  FSI
FSI
FPER
```

El que fa aquest codi és, per a cada posició del personatge principal, fa un recorregut del vector d'elements del mapa i avalua per a cada un d'ells la seva

situació. Un element en el mapa pot tenir tres situacions diferents:

- L'element no ha d'estar en pantalla → S'actualitza l'status de l'objecte i es borra de la pantalla.
- L'element ha d'estar en pantalla i no està mostrat → S'actualitza l'status de l'objecte i es mostra en pantalla.
- L'element ha d'estar en pantalla i està mostrat → Avaluem si el personatge principal li provoca algun esdeveniment.

En el cas de que el personatge principal li provoqui un esdeveniment, avaluem quin tipus d'objecte és i en base a això, l'objecte executarà una acció prèviament definida.

En la següent figura es pot veure gràficament el diagrama d'activitat associat al pseudocodi.

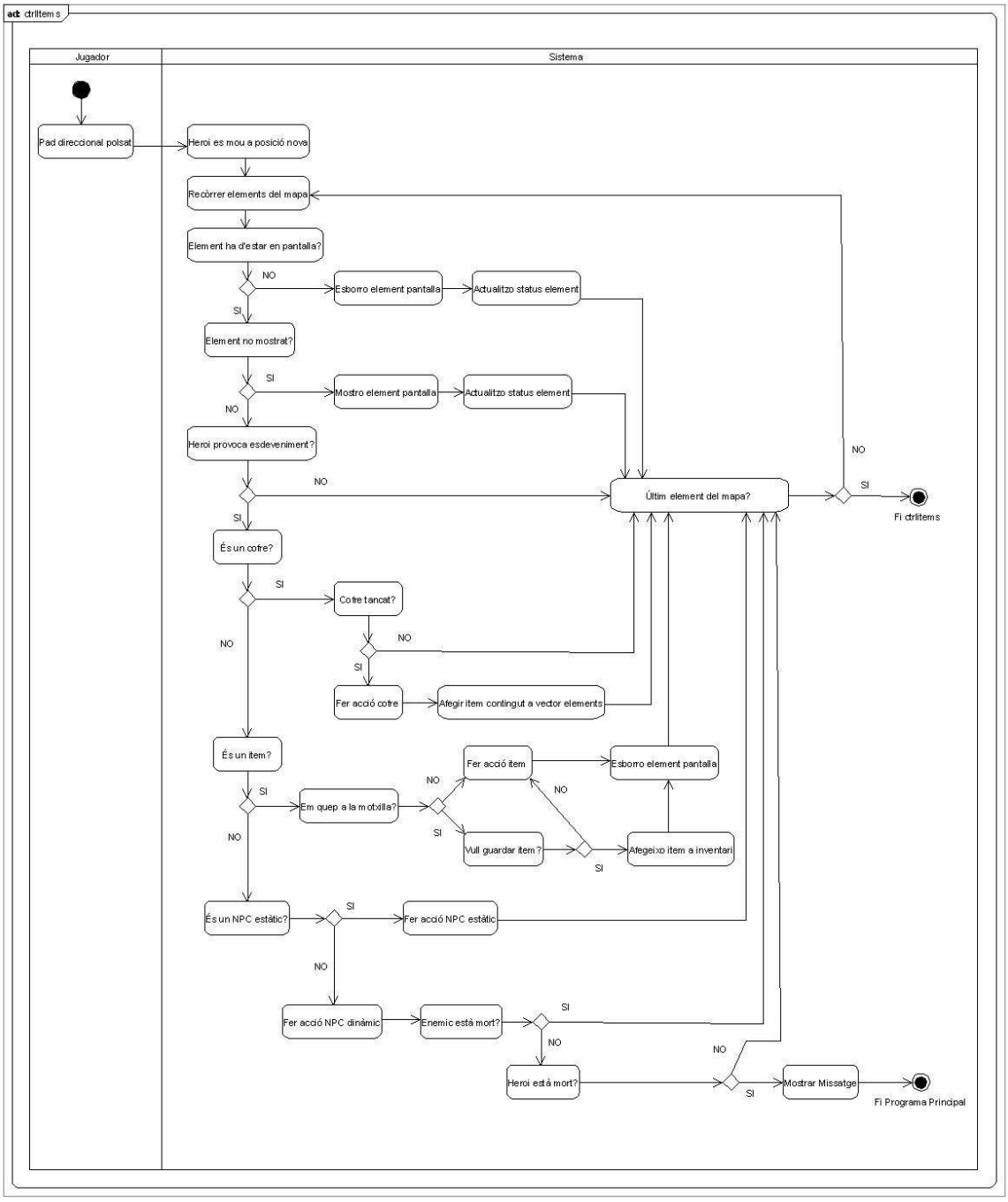


Figura 34: Diagrama d'activitat de ctrlItems.

8.5.2 Classe Mapa

La classe Mapa és la encarregada de guardar els atributs físics del mapa, com poden ser les mides del mapa i les posicions per a poder fer l'scrolling. També conté un vector on es guarden tots els objectes de tipus *Element* que podran interactuar amb l'objecte de tipus *Hero*.

Mapa
+ positionX: s32 + positionY: s32 + nFondo: s32 + limitX: s32 + limitY: s32 + vectorElement: Element + nElem: s32
+ Mapa(s32 numFondo, s32 tamanyX, s32 tamanyY, s32 posIniX, s32 posIniY) + setPositionX(s32 x) + setPositionY(s32 y) + setNFondo(s32 n) + setLimitX(s32 x) + setLimitY(s32 y) + s32 getPositionX() + s32 getPositionY() + s32 getNFondo() + s32 getLimitX() + s32 getLimitY() + setNElem(s32 x) + s32 getNElem() + putElement(Element *e) + updateElement(Element *e, s32 x) + Element * getElement(s32 x) + dropElement(s32 x) + bool elementInScreen(Element *e)

Atributs:

+ positionX: Variable de tipus enter de 32 bits, defineix la posició inicial X del mapa a la pantalla.

+ positionY: Variable de tipus enter de 32 bits, defineix la posició inicial Y del mapa a la pantalla.

+ nFondo: Variable de tipus enter de 32 bits, defineix el nivell de profunditat del

mapa.

+ limitX: Variable de tipus enter de 32 bits, defineix la longitud X màxima (en píxels).

+ limitY: Variable de tipus enter de 32 bits, defineix la longitud Y màxima (en píxels).

+ vectorElement: Vector d'objectes de tipus Element, conté tots els elements (Items, NPC i Chest) que conté el mapa.

+ nElem: Variable de tipus enter de 32 bits, defineix la quantitat d'objectes que conté el vectorElement.

Mètodes:

+ Mapa(s32 numFondo, s32 tamanyX, s32 tamanyY, s32 posIniX, s32 posIniY): Constructor de la classe Mapa, inicialitza els atributs necessaris per a poder instanciar un objecte de la classe Mapa.

+ setPositionX(s32 x): Mètode que modifica la variable positionX.

+ setPositionY(s32 y): Mètode que modifica la variable positionY.

+ setNFondo(s32 n): Mètode que modifica la variable nFondo.

+ setLimitX(s32 x): Mètode que modifica la variable limitX.

+ setLimitY(s32 y): Mètode que modifica la variable limitY.

+ s32 getPositionX(): Mètode que retorna el valor de la variable positionX.

+ s32 getPositionY(): Mètode que retorna el valor de la variable positionY.

+ s32 getNFondo(): Mètode que retorna el valor de la variable nFondo.

+ s32 getLimitX(): Mètode que retorna el valor de la variable limitX.

+ s32 getLimitY(): Mètode que retorna el valor de la variable limitY.

+ setNElem(s32 x): Mètode que modifica la variable nElem.

+ s32 getNElem(): Mètode que retorna el valor de la variable nElem.

+ putElement(Element *e): Mètode que afegeix al vector vectorElement un objecte del tipus element.

+ updateElement(Element *e, s32 x): Mètode que insereix en una posició determinada del vectorElement, un objecte del tipus Element.

+ Element * getElement(s32 x): Mètode que retorna l'objecte de tipus Element que ocupa una determinada posició al vectorElement.

+ dropElement(s32 x): Mètode que elimina l'objecte de tipus Element que ocupa una posició determinada al vectorElement.

El pseudocodi d'aquest mètode és el següent:

```
PER i = x FINS nElem-1 PAS 1 FER
  vectorElement[i] = vectorElement[i+1];
FPER
Decremento_nElem
```

Aquest mètode reordena el vector eliminant l'element que ocupa la posició X. Un cop reordenat decrementa la variable que indica el número d'elements del vector.

+ bool elementInScreen(Element *e): Mètode que retorna un booleà indicant si l'objecte de tipus Element ha d'estar mostrat en pantalla o no.

El pseudocodi d'aquest mètode és el següent:

```
limitX, limitY: S32
inScreen: bool
limitX = posicióMapaX + MàximPantallaX
limitY = posicióIMapaY + MàximPantallaY
inScreen = Fals

SI posicióItemX ENTRE (posicióMapaX, limitX) I
posicióItemY ENTRE (posicióMapaY, limitY) LLAVORS
  inScreen : Cert
FSI
RETORNA inScreen
```

Aquest mètode avalua si l'element està posicionat en pantalla i retorna un booleà indicant si es Cert o Fals.

8.5.3 Classe Hero

La classe Hero implementa l'heroi del videojoc. Aquesta classe conté tots els atributs de l'heroi, com són les posicions relatives en el mapa, les posicions en pantalla, el nivell de vida que té i diferents atributs referents a la seva representació gràfica en

pantalla.

Hero
+ positionX: s32 + positionY: s32 + positionMapX: s32 + positionMapY: s32 + mov: u16 + altitude: u16 + life: u16 + nSprite: s32 + nPaleta: s32 + bg: Bag
+ Hero(s32 numSprites, s32 numPaleta, u16 q_life) + setPositionX(s32 x) + setPositionY(s32 y) + setNSprite(s32 n) + setMov(u16 b) + setAltitude(u16 x) + s32 getPositionX() + s32 getPositionY() + s32 getNSprite() + u16 getMov() + u16 getAltitude() + setPositionMapX(s32 x) + setPositionMapY(s32 y) + s32 getPositionMapX() + s32 getPositionMapY() + setNPaleta(s32 x) + s32 getNpaleta() + setLife(u16 x) + u16 getLife() + putItemBag(Item * it) + dropItemBag(u16 x) + u16 getVolFreeBag()

+ bool isNotFullBag() + s32 itemsInBag() + Item * itemBag(u16 i)

Atributs:

+ positionX: Variable de tipus enter de 32 bits, defineix la posició X de l'heroi a la pantalla.

+ positionY: Variable de tipus enter de 32 bits, defineix la posició Y de l'heroi a la pantalla.

+ positionMapX: Variable de tipus enter de 32 bits, defineix la posició X de l'heroi relativa al mapa.

+ positionMapY: Variable de tipus enter de 32 bits, defineix la posició Y de l'heroi relativa al mapa.

+ mov: Variable de tipus real de 16 bits, defineix el número de frame associat al gràfic mostrat en pantalla de l'heroi.

+ altitude: Variable de tipus real de 16 bits, defineix el nivell on es troba l'heroi.

+ life: Variable de tipus real de 16 bits, defineix la quantitat de vida de l'heroi.

+ nSprite: Variable de tipus enter de 32 bits, defineix el número d'sprite de l'heroi.

+ nPaleta: Variable de tipus enter de 32 bits, defineix la paleta de l'heroi.

+ bg: Variable de tipus Bag on es guarden els elements de tipus Item que l'heroi agafa.

Mètodes:

+ Hero(s32 numSprites, s32 numPaleta, u16 q_life): Constructor de la classe Hero, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Hero.

+ setPositionX(s32 x): Mètode que modifica la variable positionX.

+ setPositionY(s32 y): Mètode que modifica la variable positionY.

+ setNSprite(s32 n): Mètode que modifica la variable nSprite.

+ setMov(u16 b): Mètode que modifica la variable mov.

+ setAltitude(u16 x): Mètode que modifica la variable altitude.

+ s32 getPositionX(): Mètode que retorna el valor de la variable positionX.

+ s32 getPositionY(): Mètode que retorna el valor de la variable positionY.

+ s32 getNSprite(): Mètode que retorna el valor de la variable nSprite.

+ u16 getMov(): Mètode que retorna el valor de la variable mov.

+ u16 getAltitude(): Mètode que retorna el valor de la variable altitude.

+ setPositionMapX(s32 x): Mètode que modifica la variable positionMapX.

+ setPositionMapY(s32 y): Mètode que modifica la variable positionMapY.

+ s32 getPositionMapX(): Mètode que retorna el valor de la variable positionMapX.

+ s32 getPositionMapY(): Mètode que retorna el valor de la variable positionMapY.

+ setNPaleta(s32 x): Mètode que modifica la variable nPaleta.

+ s32 getNpaleta(): Mètode que retorna el valor de la variable nPaleta.

+ setLife(u16 x): Mètode que modifica la variable life.

+ u16 getLife(): Mètode que retorna el valor de la variable life.

+ putItemBag(Item * it): Mètode que afegeix un objecte de tipus Item a la motxilla.

+ dropItemBag(u16 x): Mètode que elimina de la motxilla, l'objecte Item que ocupa la posició x.

+ u16 getVolFreeBag(): Mètode que retorna la quantitat de volum lliure de la motxilla.

+ bool isNotFullBag(): Mètode que retorna un booleà indicant si la motxilla està plena o no.

+ s32 itemsInBag(): Mètode que retorna el número d'elements que hi ha a la motxilla de l'heroi.

+ Item * itemBag(u16 i): Mètode que retorna de la motxilla, l'objecte de tipus Item, que ocupa la posició i.

8.5.4 Classe Bag

La classe Bag és una classe contenidora d'elements de tipus Item. Està definida com a un vector d'Items. Aquesta classe conté els atributs necessaris per a poder tractar i consultar el vector d'Items.

Bag
+ nElem: u16 + vol: u16 + volMax: u16 + maxElem: u16 + vectorItems[10]: Item
+ Bag() + setNElem(u16 x) + u16 getNElem() + u16 getMaxElem() + bool isNotFull() + putItem(Item *it) + Item *getItem(u16 x) + dropItem(u16 x) + u16 getVolFree()

Atributs:

+ nElem: Variable de tipus real de 16 bits, defineix el número d'elements que hi ha al vector.

+ vol: Variable de tipus real de 16 bits, defineix el volum total ocupat a la motxilla.

+ volMax: Variable de tipus real de 16 bits, defineix el volum màxim que pot tenir la motxilla.

+ maxElem: Variable de tipus real de 16 bits, defineix el número màxim d'elements que hi poden haver a la motxilla.

+ vectorItems[10]: Variable de tipus vector d'Items on es guardaran tots els Items.

Mètodes:

+ Bag(): Constructor de la classe Bag, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Bag.

+ setNElem(u16 x): Mètode que modifica la variable nElem.

+ u16 getNElem(): Mètode que retorna el valor de la variable nElem.

+ u16 getMaxElem(): Mètode que retorna el valor de la variable maxElem.

+ bool isNotFull(): Mètode que retorna cert o fals, depenent si el valor nElem es igual al valor de maxElem, respectivament.

+ putItem(Item *it): Mètode que afegeix al vector vectorItems[], l'element it de tipus Item (passat per paràmetre) .

+ Item *getItem(u16 x): Mètode que retorna l'objecte Item que ocupa la posició x del vector d'Items.

+ dropItem(u16 x): Mètode que elimina l'objecte Item que ocupa la posició x del vector d'Items.

+ u16 getVolFree(): Mètode que retorna la diferencia entre les variables volMax i vol.

8.5.5 Classe Element

La classe Element és una classe de tipus genèrica i és pare de les classes Npc, Chest i Item. Aquesta classe serveix per a representar gràficament tots els elements estàtics i dinàmics que conté el mapa.

Element
+ positionX: s32 + positionY: s32 + positionScreenX: s32 + positionScreenY: s32 + nSprite: s32 + nPaleta: s32 + GFX: u16 + nImage: u16

+ Visible: bool + ev: Condition + act: Condition
+ Element(s32 x, s32 y, s32 d, u16 image, u16 key) + Element(Element * e) + virtual ~Element() + s32 getPositionX() + s32 getPositionY() + setPositionX(s32 x) + setPositionY(s32 y) + setPositionScreenX(s32 x) + setPositionScreenY(s32 y) + s32 getPositionScreenX() + s32 getPositionScreenY() + setNSprite(s32 n) + s32 getNSprite() + setGFX(u16 num) + u16 getGFX() + setNPaleta(s32 x) + s32 getNpaleta() + setVisible() + setInvisible() + bool getVisible() + bool evaluate(Hero *p, u16 key) + u16 getNImage() + setNImage(u16 x)

Atributs:

+ positionX: Variable de tipus enter de 32 bits, defineix la posició X de l'objecte Element en el mapa.

+ positionY: Variable de tipus enter de 32 bits, defineix la posició Y de l'objecte Element en el mapa.

+ positionScreenX: Variable de tipus enter de 32 bits, defineix la posició X de l'objecte Element a la pantalla.

+ positionScreenY: Variable de tipus enter de 32 bits, defineix la posició Y de

l'objecte Element a la pantalla.

+ nSprite: Variable de tipus enter de 32 bits, defineix el número de l'sprite de l'objecte Element.

+ nPaleta: Variable de tipus enter de 32 bits, defineix la paleta de l'objecte Element.

+ GFX: Variable de tipus real de 16 bits, defineix un valor que permetrà accedir a la imatge de l'objecte Element per aquest valor.

+ nImage: Variable de tipus real de 16 bits, defineix el frame de l'sprite de l'objecte Element.

+ Visible: Variable de tipus booleà, defineix si l'objecte Element s'està mostrant en pantalla.

+ ev: Variable de tipus Condition, aquesta conté les condicions que s'han de complir per a provocar un esdeveniment a l'objecte de tipus Element.

+ act: Variable de tipus Condition, aquesta conté les accions que farà l'objecte de tipus Element si se li provoca un esdeveniment.

Mètodes:

+ Element(s32 x, s32 y, s32 d, u16 image, u16 key)): Constructor de la classe Element, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Element.

+ Element(Element * e): Constructor còpia, crea una instància igual a la l'objecte passat per paràmetre.

+ virtual ~Element(): Mètode destructor virtual.

+ s32 getPositionX(): Mètode que retorna el valor de la variable positionX.

+ s32 getPositionY(): Mètode que retorna el valor de la variable positionY.

+ setPositionX(s32 x): Mètode que modifica el valor de la variable positionX.

El pseudocodi d'aquest mètode és el següent:

```
positionX=x
ic: isClose
ic = ev.getCondition()
 ic.setPosX(x)
```

Aquest mètode modifica la posició X del mapa amb el valor de la variable X passada per paràmetre. També modifica la posició X del mapa guardada a la classe isClose.

+ setPositionY(s32 y): Mètode que modifica el valor de la variable positionY.

El pseudocodi d'aquest mètode és el següent:

```
positionY=y
ic: isClose
ic = ev.getCondition()
ic. setPosY(y)
```

Aquest mètode modifica la posició Y del mapa amb el valor de la variable Y passada per paràmetre. També modifica la posició Y del mapa guardada a la classe isClose.

+ setPositionScreenX(s32 x): Mètode que modifica el valor de la variable positionScreenX.

+ setPositionScreenY(s32 y): Mètode que modifica el valor de la variable positionScreenY.

+ s32 getPositionScreenX(): Mètode que retorna el valor de la variable positionScreenX.

+ s32 getPositionScreenY(): Mètode que retorna el valor de la variable positionScreenY.

+ setNSprite(s32 n): Mètode que modifica el valor de la variable nSprite.

+ s32 getNSprite(): Mètode que retorna el valor de la variable nSprite.

+ setGFX(u16 num): Mètode que modifica el valor de la variable GFX.

+ u16 getGFX(): Mètode que retorna el valor de la variable GFX.

+ setNPaleta(s32 x): Mètode que modifica el valor de la variable nPaleta.

+ s32 getNpaleta(): Mètode que retorna el valor de la variable nPaleta.

+ setVisible(): Mètode que modifica el valor de la variable Visible.

+ setInvisible(): Mètode que modifica el valor de la variable Visible.

+ bool getVisible(): Mètode que retorna el valor de la variable Visible.

+ bool evaluate(Hero *p, u16 key): Mètode que avalua si l'ítem està prou a prop i la tecla polsada li provoca algun esdeveniment.

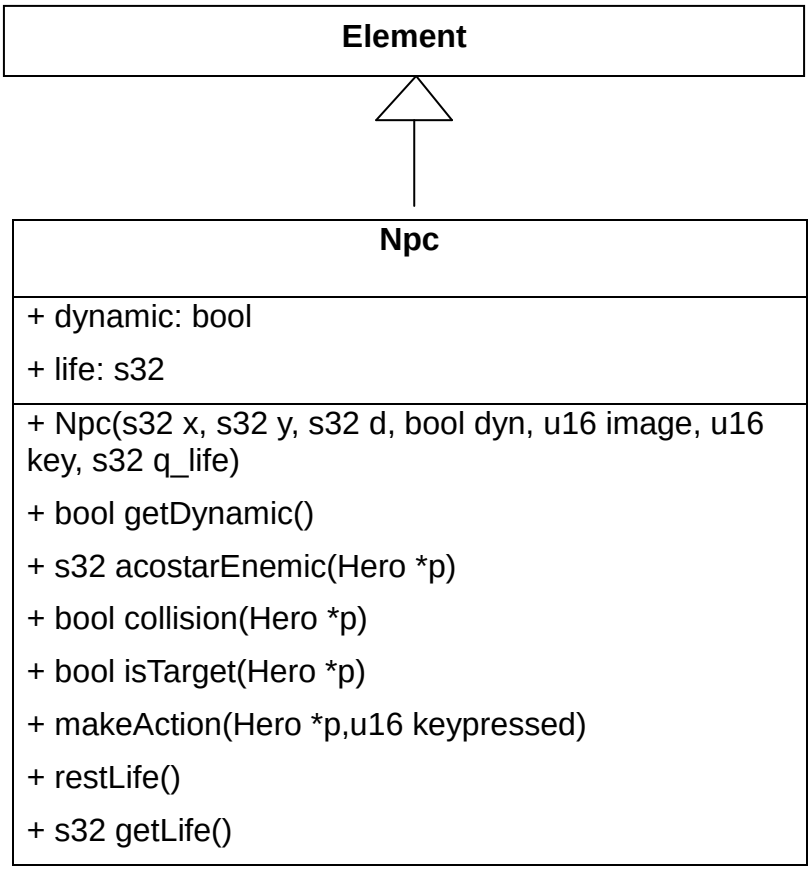
+ u16 getNImage(): Mètode que retorna el valor de la variable nImage.

+ setNImage(u16 x): Mètode que modifica el valor de la variable nImage.

+ Condition * getCondition(): Mètode que retorna la referencia de l'objecte Condition, de la variable ev.

8.5.6 Classe Npc

La classe Npc hereta de la classe Element. Els objectes d'aquesta classe representen tant els personatges enemics com els personatges NPC. Els personatges NPC són elements estàtics i els elements enemics són elements dinàmics.



Atributs:

+ dynamic: Variable de tipus booleà, defineix si l'objecte Npc és un NPC (element estàtic en el mapa) o un enemic (element dinàmic en el mapa).

+ life: Variable de tipus enter de 32 bits, defineix la quantitat de vida de l'objecte

Npc.

Mètodes:

+ Npc(s32 x, s32 y, s32 d, bool dyn, u16 image, u16 key, s32 q_life): Constructor de la classe Npc, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Npc.

+ bool getDynamic(): Mètode que retorna el valor de la variable dynamic.

+ s32 acostarEnemic(Hero *p): Mètode que modifica les posicions de l'objecte Npc, per tal que s'acosti a la posició de l'objecte Hero. Retorna un valor numèric per indicar si s'ha acostat per l'esquerra (1) o per la dreta (2).

El pseudocodi d'aquest mètode és el següent:

```
incX, incY, result: s32
incX = 0
incY = 0
result = 0


SI posicióPersonatgeMapaY + migPantallaY > posicióNPCY
LLAVORS
  incY = 1
ALTRAMENT SI posicióPersonatgeMapaY + migPantallaY < posicióNPCY
  incY = -1
ALTRAMENT
  SI posicióPersonatgeMapaX+migPantallaX > posicióNPCX+16 LLAVORS
    incX = 1;
    result = 1;
  ALTRAMENT SI posicióPersonatgeMapaY+migPantallaY > posicióNPCY-16 LLAVORS
    incX = -1;
    result = 2;
  FSI
FSI
actualitzaPosicióNPCX(incX)
actualitzaPosicióNPCY(incY)


RETORNA result
```

Aquest mètode acosta la posició de l'objecte Npc a la posició de l'objecte Hero. La

primera aproximació que fa és una aproximació vertical. Un cop s'ha alineat per la ordenada y amb l'objecte Hero, s'aproxima horitzontalment fins a la posició de l'objecte Hero, fins a provocar una col·lisió. Un cop acabada l'execució, el mètode retorna un valor enter per a informar per a quin costat s'ha aproximat esquerra (1) o dreta (2). Aquest mètode implementa la IA (intel·ligència artificial) de l'enemic, és un codi que es pot modificar per altres algoritmes de IA més eficients.

+ bool collision(Hero *p): Mètode que avalua si l'objecte NPC està col·lidint lateralment amb l'objecte Hero.

+ bool isTarget(Hero *p): Mètode que avalua si l'objecte NPC pot ser atacat per l'objecte Hero.

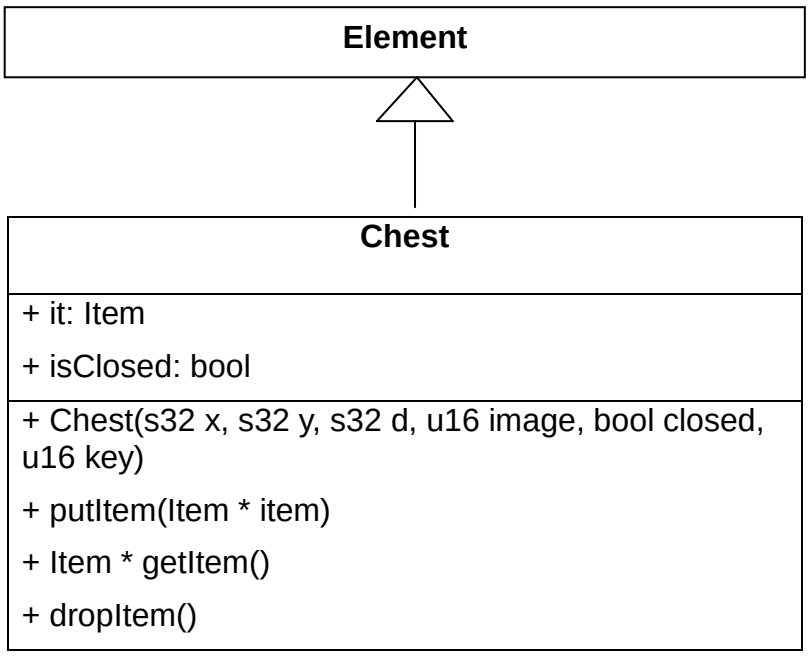
+ makeAction(Hero *p, u16 keypressed): Mètode que executa la funció evaluate(Hero *p, u16 keypressed) de la variable act, de tipus action.

+ restLife(): Mètode que modifica el valor de la variable life. El nou valor es calcula restant a la variable life el valor del paràmetre life_atac, definit a la classe Defines.

+ s32 getLife(): Mètode que retorna el valor de la variable life.

8.5.7 Classe Chest

La classe Chest hereta de la classe Element. Les instàncies d'aquesta classe són objectes contenidors que poden contenir un objecte del tipus Item.



+ setClosed(bool b) + bool getClosed() + open() + makeAction(Hero *p,u16 keypressed)

Atributs:

+ it: Variable de tipus Item, defineix l'objecte Item que contindrà l'objecte Chest.

+ isClosed: Variable de tipus booleà, defineix si el cofre està obert o tancat.

Mètodes:

+ Chest(s32 x, s32 y, s32 d, u16 image, bool closed, u16 key): Constructor de la classe Chest, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Chest.

+ putItem(Item * item): Mètode que afegeix un objecte Item.

+ Item * getItem(): Mètode que retorna l'objecte que conté l'objecte Chest.

+ dropItem(): Mètode que elimina l'objecte Item que conté l'objecte Chest.

+ setClosed(bool b): Mètode que modifica el valor de la variable isClosed.

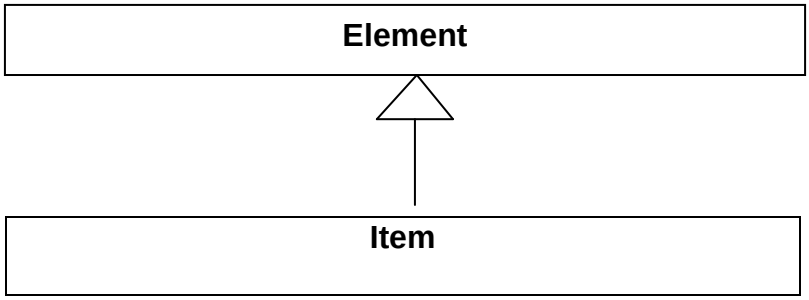
+ bool getClosed(): Mètode que retorna el valor de la variable isClosed.

+ open(): Mètode que modifica el valor de la variable isClosed a cert.

+ makeAction(Hero *p, u16 keypressed): Mètode que executa la funció evaluate(Hero *p, u16 keypressed) de la variable act, de tipus action.

8.5.8 Classe Item

La classe Item hereta de la classe Element. Els objectes d'aquesta classe són elements del mapa que poden ser guardats o utilitzats per l'objecte Hero.



+ volum: s32
+ Item(s32 x, s32 y, s32 d, u16 image, s32 vol, bool v, u16 key) + setVol(s32 vol) + s32 getVol()

Atributs:

+ volum: Variable de tipus enter de 32 bits, defineix el volum que ocupa l'objecte de tipus Item.

Mètodes:

+ Item(s32 x, s32 y, s32 d, u16 image, s32 vol, bool v, u16 key): Constructor de la classe Item, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Item.

+ setVol(s32 vol): Mètode que modifica el valor de la variable vol.

+ s32 getVol(): Mètode que retorna el valor de la variable volum.

+ makeAction(Hero *p, u16 keypressed): Mètode que executa la funció evaluate(Hero *p, u16 keypressed) de la variable act, de tipus action.

8.5.9 Classe Condition

La classe Condition és una classe abstracta i és pare de les classes isClose, Event i Action.

Condition
+ virtual ~Condition() = 0 + virtual bool evaluate(Hero *p, u16 key, Element *e) = 0

Atributs:

Aquesta classe no té atributs.

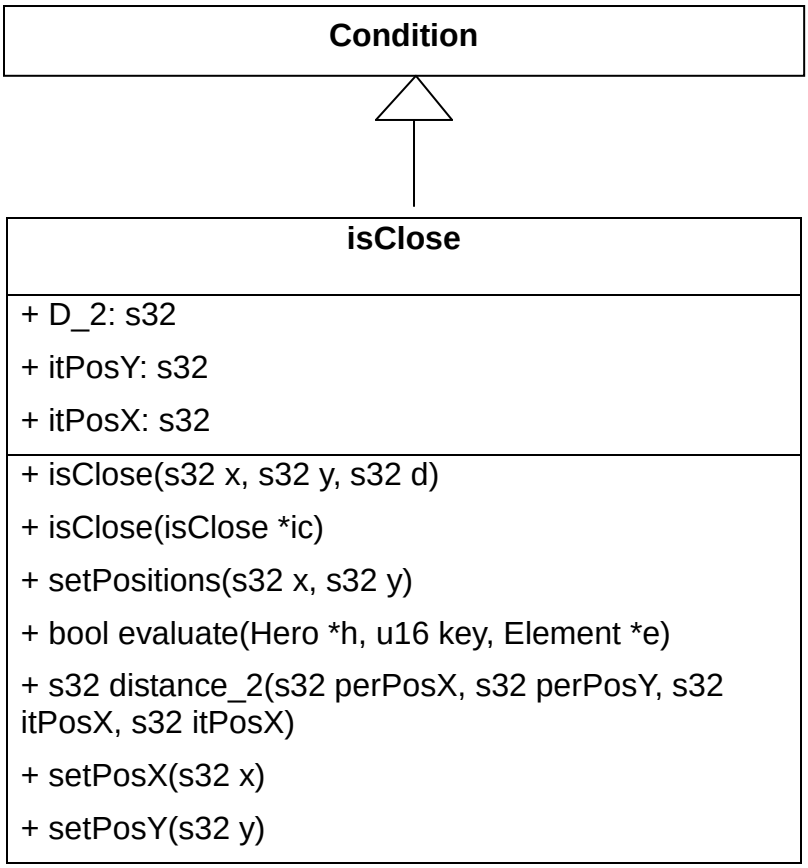
Mètodes:

virtual ~Condition() = 0: Destructor virtual de la classe Condition.

virtual bool evaluate(Hero *p, u16 key, Element *e) = 0: Funció virtual de la classe Condition.

8.5.10 Classe isClose

La classe isClose hereta de la classe Condition. Aquesta classe avalua si la distància entre l'objecte Hero i l'objecte Element sigui igual o menor a la predefinida.



Atributs:

+ D_2: Variable de tipus enter de 32 bits, defineix el quadrat de la distància en la que l'objecte de tipus Hero, hi pot interactuar.

+ itPosY: Variable de tipus enter de 32 bits, defineix la posició y de l'item en el mapa.

+ itPosX: Variable de tipus enter de 32 bits, defineix la posició x de l'item en el mapa.

Mètodes:

+ isClose(s32 x, s32 y, s32 d): Constructor de la classe isClose, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe isClose.

+ isClose(isClose *ic): Constructor còpia, crea una instància igual a la l'objecte passat per paràmetre.

+ setPositions(s32 x, s32 y): Mètode que modifica el valor de la variable itPosX i itPosY.

+ bool evaluate(Hero *h, u16 key, Element *e): Mètode que retorna cert si la distància al quadrat entre l'objecte Hero i la posició de l'objecte Element és mes petita o igual que la distància al quadrat D_2.

+ s32 distance_2(s32 perPosX, s32 perPosY, s32 itPosX, s32 itPosX): Mètode que retorna el càlcul del valor de la distància al quadrat, entre la posició de l'objecte Hero i la posició de l'objecte Element.

El pseudocodi d'aquest mètode és el següent:

```
x, y: s32
x = posicióHeroX + MigPantallaX - posicióItemX
y = posicióHeroY + MigPantallaY - posicióItemY
RETORNA x * x + y * y
```

Com es pot veure primer es calcula la distància entre coordenades i després es calcula la distància lineal al quadrat. A l'hora de treballar amb distàncies és treballa amb valors al quadrat ja que per motius de rendiment és més fàcil treballar amb valors quadrats que no haver de calcular l'arrel quadrada corresponent.

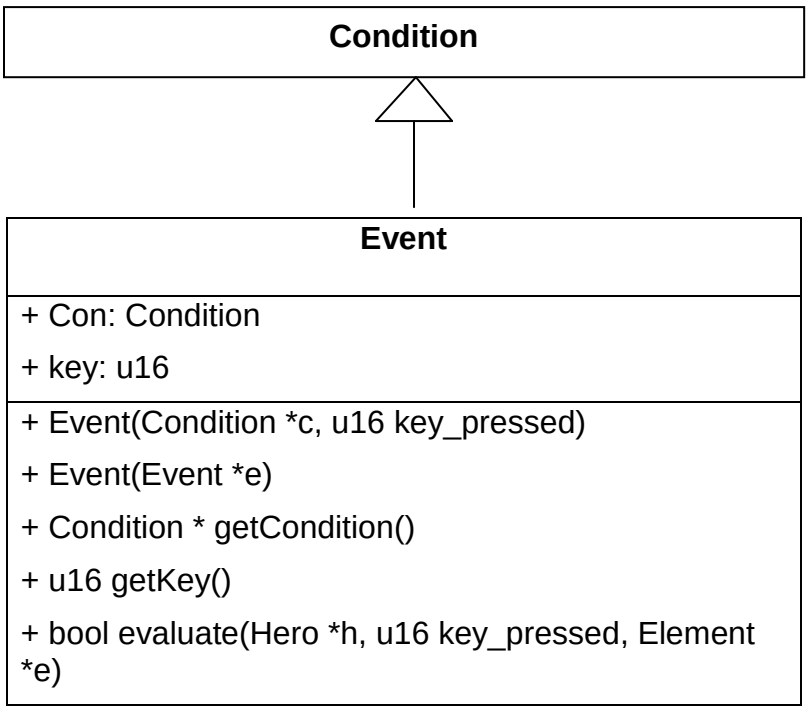
+ setPosX(s32 x): Mètode que modifica el valor de la variable itPosX.

+ setPosY(s32 y): Mètode que modifica el valor de la variable itPosY.

8.5.11 Classe Event

La classe Event hereta de la classe Condition. Aquesta classe avalua si es compleixen totes les condicions per a que un objecte del tipus Hero provoqui un

esdeveniment a l'objecte de tipus Element. Aquesta classe conté un objecte de la tipus isClose.



Atributs:

+ Con: Variable de tipus Condition, ve definida per un objecte isClose que definirà si la distància és correcte per provocar un esdeveniment.

+ key: Variable de tipus real de 16 bits, defineix la tecla. Si el valor de la variable key és 0, llavors es defineix que no es polsa cap tecla.

Mètodes:

+ Event(Condition *c, u16 key_pressed): Constructor de la classe Event, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe Event.

+ Event(Event *e): Constructor còpia, crea una instància igual a la l'objecte passat per paràmetre.

+ Condition * getCondition(): Mètode que retorna l'objecte Con (isClose) de tipus Condition.

+ u16 getKey(): Mètode que retorna el valor de la variable key.

+ bool evaluate(Hero *h, u16 key_pressed, Element *e): Mètode que retorna

cert si es compleixen totes les condicions per a provocar un esdeveniment l'objecte Element.

El pseudocodi d'aquest mètode és el següent:

```
RETORNA avaluarDistanciaIsClose I key_pressed == key
```

En la següent figura es pot veure gràficament el diagrama de seqüència associat al pseudocodi.

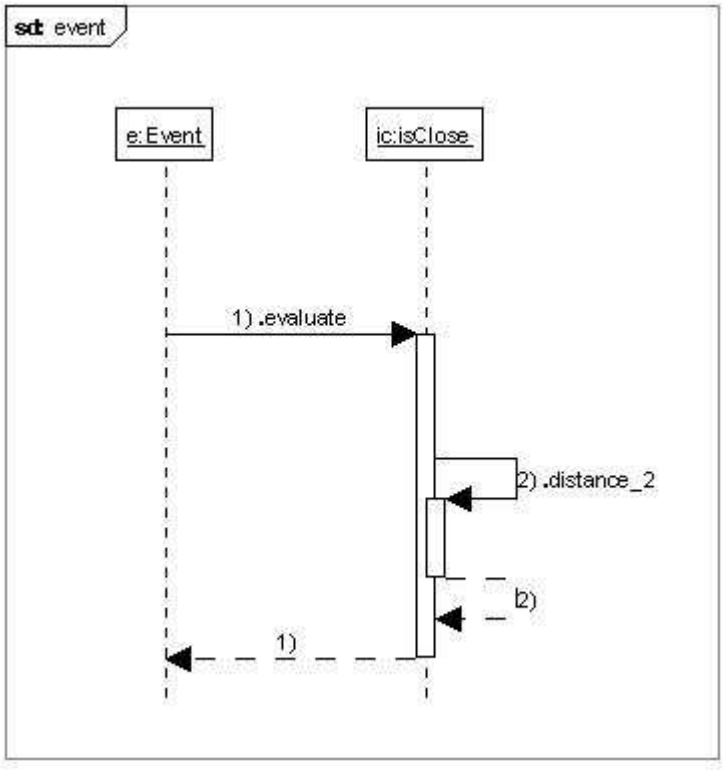
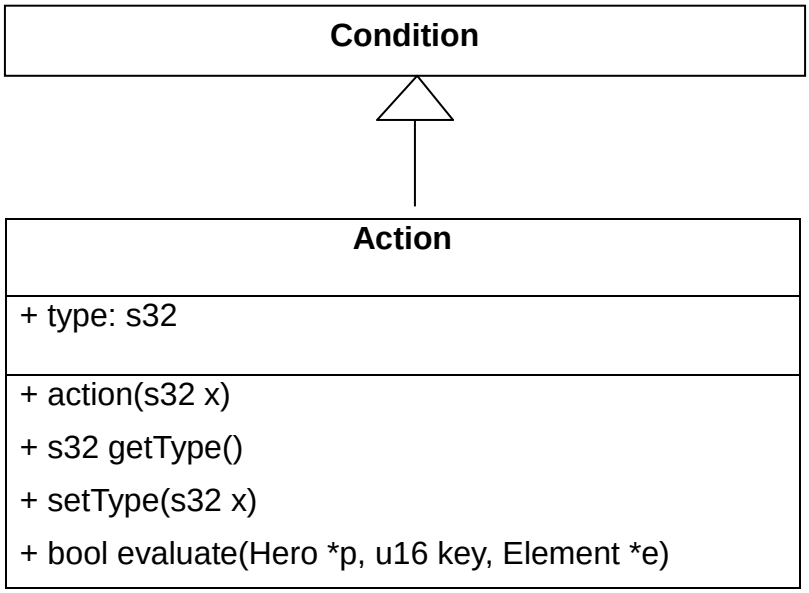


Figura 34: Diagrama de seqüència del mètode evaluate de la classe Event.

8.5.12 Classe action

La classe action hereta de la classe Condition. Els objectes d'aquesta classe executen les accions predefinides dels objectes de tipus Element.



Atributs:

+ type: Variable de tipus enter de 32 bits, defineix el tipus d'acció:

Type: 0 -> Item, 1 -> NPC, 2 -> Enemy, 3 -> Chest

Mètodes:

+ action(s32 x): Constructor de la classe action, encarregat d'inicialitzar tots els atributs necessaris per a poder instanciar un objecte de la classe action.

+ s32 getType(): Mètode que retorna el valor de la variable type.

+ setType(s32 x): Mètode que modifica el valor de la variable type.

+ bool evaluate(Hero *p, u16 key, Element *e): Mètode que executa les accions definides per l'objecte Element, depenent de la variable type.

El pseudocodi d'aquest mètode és el següent:

```
AVALUA type
  CAS 0 LLAVORS
    augmentarVidaHeroi
  FCAS
  CAS 1 LLAVORS
    mostrarDiàlegNPC
  FCAS
  CAS 2:
    acostarEnemicAHeroi
    SI heroiAtacaEnemic LLAVORS
      restarVidaEnemic
```

```
    ALTRAMENT enemicAtacaHeroi LLAVORS
      restarVidaHeroi
    FSI
  FCAS
  CAS 3 LLAVORS
    ObrirCofre
    cambiarImatgeCofreObert
  FCAS
FAVALUA
```

Aquest codi executa una sèrie d'accions depenent del valor da la variable type. Hi ha definit 4 tipus d'accions diferents, que es corresponen (per ordre) a si l'Element és una poció, un NPC, un enemic o un cofre.

En el CAS 0 l'algoritme crida a la funció setLife(u16 x) de la classe Hero, passant com a paràmetre X la vida actual de l'heroi més el paràmetre life_Potion de la classe Defines.

En el CAS 1 l'algoritme mostra una sèrie de missatges predefinits per pantalla, on es mostra el diàleg de l'objecte Npc.

En el CAS 2 l'algoritme crida a la funció acostarEnemic de la classe Npc. Després s'avalua si hi ha hagut una col·lisió (es resta la vida a l'heroi) o l'objecte heroi li ha provocat un esdeveniment abans de col·lidir (es resta la vida a l'enemic).

En el CAS 3 es canvia la imatge l'status del cofre de tancat a obert i es modifica la imatge.

8.5.13 Classe Defines

Aquesta classe de definició conté tots els paràmetres configurables i no configurables del motor de joc.

Defines
+ migPantallaX: s32 + migPantallaY: s32 + ScreenDown: u16 + ScreenUp: u16 + PantallaX: s32

+ PantallaY: s32
+ tamanyX: s32
+ tamanyY: s32
+ tamanyXTiles: s32
+ tamanyYTiles: s32
+ distItem: s32
+ distMonster: s32
+ distPNC: s32
+ distAtacHero: s32
+ itemStatic: bool
+ itemDynamic: bool
+ chest_closed: u16
+ chest_opened: u16
+ Key: u16
+ Potion: u16
+ non_player_char: u16
+ vol_key: u16
+ vol_potion: u16
+ no_key_pressed: u16
+ key_pressed_A: u16
+ key_pressed_B: u16
+ key_pressed_X: u16
+ key_pressed_Y: u16
+ key_pressed_Start: u16
+ life_Hero: u16
+ life_Monster: u16
+ life_NPC: u16
+ life_Potion: u16
+ life_atac: u16
+ type_Potion: u16
+ type_NPC: u16
+ type_Enemy: u16
+ type_Chest: u16

Paràmetres:

+ migPantallaX: Constant de tipus enter de 32 bits, defineix la coordenada X (en

píxels) de la posició mitja de la pantalla. Ve definida per el hardware de la consola.

+ migPantallaY: Constant de tipus enter de 32 bits, defineix la coordenada Y (en píxels) de la posició mitja de la pantalla. Ve definida per el hardware de la consola.

+ ScreenDown: Constant de tipus real de 16 bits, defineix quina és la pantalla inferior.

+ ScreenUp: Constant de tipus real de 16 bits, defineix quina és la pantalla superior.

+ PantallaX: Constant de tipus enter de 32 bits, defineix la longitud X (en píxels) de la pantalla. Ve definida per el hardware de la consola.

+ PantallaY: Constant de tipus enter de 32 bits, defineix la longitud Y (en píxels) de la pantalla. Ve definida per el hardware de la consola.

+ tamanyX: Variable de tipus enter de 32 bits, defineix la longitud X (en píxels) del mapa. Varia en funció del mapa utilitzat.

+ tamanyY: Variable de tipus enter de 32 bits, defineix la longitud Y (en píxels) del mapa. Varia en funció del mapa utilitzat.

+ tamanyXTiles: Variable de tipus enter de 32 bits, defineix la mida del mapa en tiles. Es calcula en funció de **tamanyX**.

+ tamanyYTiles: Variable de tipus enter de 32 bits, defineix la mida del mapa en tiles. Es calcula en funció de **tamanyY**.

+ distItem: Constant de tipus enter de 32 bits, defineix la distància màxima per a poder interactuar amb l'element.

+ distMonster: Constant de tipus enter de 32 bits, defineix la distància màxima per a poder interactuar amb l'element.

+ distPNC: Constant de tipus enter de 32 bits, defineix la distància màxima per a poder interactuar amb l'element.

+ distAtacHero: Constant de tipus enter de 32 bits, defineix la distància màxima per a poder atacar un enemic.

+ itemStatic: Constant de tipus booleà, defineix si un element és estàtic.

+ itemDynamic: Constant de tipus booleà, defineix si un element és dinàmic.

+ chest_closed: Constant de tipus enter de 16 bits, defineix l'sprite associat a una

imatge de cofre tancat.

+ chest_opened: Constant de tipus real de 16 bits, defineix l'sprite associat a una imatge de cofre obert.

+ Key: Constant de tipus real de 16 bits, defineix l'sprite associat a una imatge clau.

+ Potion: Constant de tipus real de 16 bits, defineix l'sprite associat a una imatge poció.

+ non_player_char: Constant de tipus real de 16 bits, defineix l'sprite associat a una imatge d'un NPC.

+ vol_key: Constant de tipus real de 16 bits, defineix el volum que ocupa una clau.

+ vol_potion: Constant de tipus real de 16 bits, defineix el volum que ocupa una poció.

+ no_key_pressed: Constant de tipus real de 16 bits, defineix la tecla associada a un esdeveniment. En aquest cas valor 0, cap tecla.

+ key_pressed_A: Constant de tipus real de 16 bits, defineix la tecla associada a un esdeveniment. En aquest cas valor 1.

+ key_pressed_B: Constant de tipus real de 16 bits, defineix la tecla associada a un esdeveniment. En aquest cas valor 2.

+ key_pressed_X: Constant de tipus real de 16 bits, defineix la tecla associada a un esdeveniment. En aquest cas valor 3.

+ key_pressed_Y: Constant de tipus real de 16 bits, defineix la tecla associada a un esdeveniment. En aquest cas 4.

+ key_pressed_Start: Constant de tipus real de 16 bits, defineix la tecla associada a un esdeveniment. En aquest cas valor 5.

+ life_Hero: Variable de tipus real de 16 bits, defineix la quantitat de vida d'un element de tipus Heroi. Ha de ser més gran que 0.

+ life_Monster: Variable de tipus real de 16 bits, defineix la quantitat de vida d'un element de tipus Enemic. Ha de ser més gran que 0.

+ life_NPC: Variable de tipus real de 16 bits, defineix la quantitat de vida d'un element de tipus NPC, per defecte 0.

+ life_Potion: Variable de tipus real de 16 bits, defineix la quantitat de vida que augmenta a l'element de tipus Heroi.

+ life_atac: Variable de tipus real de 16 bits, defineix la quantitat de vida que decrementa a l'element de tipus Heroi.

+ type_Potion: Variable de tipus real de 16 bits, defineix el tipus d'acció que executarà l'objecte de tipus Item.

+ type_NPC: Variable de tipus real de 16 bits, defineix el tipus d'acció que executarà l'objecte de tipus Item.

+ type_Enemy: Variable de tipus real de 16 bits, defineix el tipus d'acció que executarà l'objecte de tipus Item.

+ type_Chest: Variable de tipus real de 16 bits, defineix el tipus d'acció que executarà l'objecte de tipus Item.

9 Implementació i proves

En aquest apartat s'explicarà com s'ha implementat cada element del projecte, per a poder arribar al resultat final.

9.1 Menú principal

El menú principal es crida des del programa principal i mostra 2 opcions al iniciar l'aplicació (veure Figura 35).

Figura 35: Menú principal.

Aquesta pantalla ve generada per el mètode pantallaInicial(). El codi és el següent:

```
void pantallaInicial(){
    bool start = false;

    while (!start){
        PA_OutputText(ScreenDown,8,8,"Prem Start per: ");
        PA_OutputText(ScreenDown,8,9,"iniciar la partida");
        PA_OutputText(ScreenDown,8,12,"Prem Select per: ");
        PA_OutputText(ScreenDown,4,13,"saber sobre el programari");
        if(Pad.Released.Start){
            start = true;
        }
        else if(Pad.Held.Select){
            PA_ClearTextBg(ScreenDown);
            while (!Pad.Held.Select){
                PA_OutputText(ScreenDown,0,1,"MOTOR 2D PER VIDEOJOCS");
                PA_OutputText(ScreenDown,8,2,"DE PLATAFORMES MOBILS");
                PA_OutputText(ScreenDown,0,7,"Alumne: Andres Quesada Molinero");
                PA_OutputText(ScreenDown,0,9,"Tutor : Gustavo Patow");
                PA_OutputText(ScreenDown,0,10,"Departament:");
                PA_OutputText(ScreenDown,0,11,"Informàtica i");
                PA_OutputText(ScreenDown,5,12,"Matemàtica Aplicada");
                PA_OutputText(ScreenDown,1,18,"Prem Select per tornar enrere.");
            }
            PA_ClearTextBg(ScreenDown);
        }
    }
    PA_ClearTextBg(ScreenDown);
}
```

Com es pot observar en el codi, un cop mostrades les 2 opcions, si polsem la tecla Select mostra informació addicional del projecte. Si polsem Select un altre cop tornem a la pantalla principal. Per sortir de la pantalla principal i iniciar el joc hem de polsar la tecla Start.

A la figura següent es pot veure la pantalla d'informació un cop polsada la tecla Select.

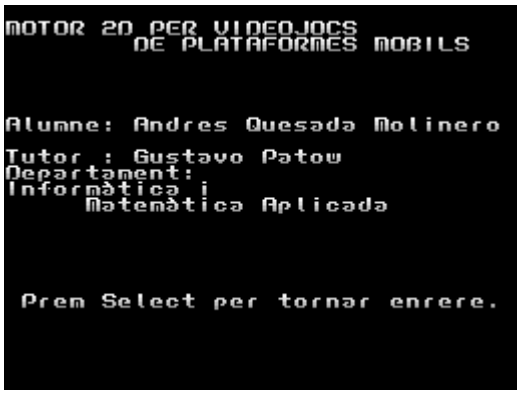


Figura 36: Pantalla informació addicional del projecte.

9.2 Scrolling

Per a realitzar l'scrolling de l'escenari s'han definit dos situacions: la primera, quan es mou el fons i l'heroi queda centrat en el mapa. La segona situació és quan la pantalla arriba a un dels extrems del mapa, llavors el mapa queda fix i es mou l'heroi.

A la Figura 37 podem veure com l'heroi està centrat a la pantalla i qui es mou és el fons.

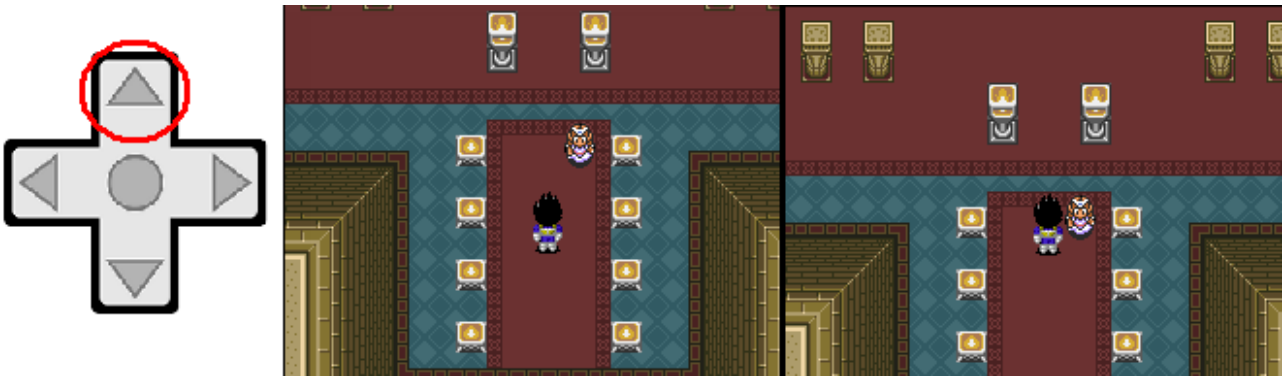


Figura 37: Exemple d'scrolling amb el moviment del fons.

A la Figura 38 podem veure com l'heroi es desplaça del centre de la pantalla i el fons queda fix.

Figura 38: Exemple d'scrolling amb el moviment de l'heroi.

El codi associat a aquesta funcionalitat és el següent:

```
void scrolling (Mapa *m, Hero *p){
    animacioBasica(p);
    if((*m).getPositionX() < (*m).getLimitX()/2){
        if((*m).getPositionX()==0){
            scrollingPersonatgeHoritzontal(m,p);
            if((*p).getPositionX()==migPantallaX ){
                (*m).setPositionX((*m).getPositionX()+1);
            }
        }
        else{
            scrollingMapaHoritzontal(m,p);
        }
    }
    else if ((*m).getPositionX() >= (*m).getLimitX()/2){
        if((*m).getPositionX()==(*m).getLimitX()){
            scrollingPersonatgeHoritzontal(m,p);
            if((*p).getPositionX()==migPantallaX ){
                (*m).setPositionX((*m).getPositionX()-1);
            }
        }
        else{
            scrollingMapaHoritzontal(m,p);
        }
    }

    if((*m).getPositionY() < (*m).getLimitY()/2){
        if((*m).getPositionY()==0){
            scrollingPersonatgeVertical(m,p);
            if((*p).getPositionY()==migPantallaY ){
                (*m).setPositionY((*m).getPositionY()+1);
            }
        }
        else{
            scrollingMapaVertical(m,p);
        }
    }
    else if ((*m).getPositionY() >= (*m).getLimitY()/2){
        if((*m).getPositionY()==(*m).getLimitY()){
            scrollingPersonatgeVertical(m,p);
            if((*p).getPositionY()==migPantallaY ){
                (*m).setPositionY((*m).getPositionY()-1);
            }
        }
        else{
            scrollingMapaVertical(m,p);
        }
    }
    (*p).setPositionMapX((*m).getPositionX());
    (*p).setPositionMapY((*m).getPositionY());

}
```

En el codi anterior podem veure com el mètode arrenca l'animació del personatge

principal i tot seguit avalua a quina meitat del mapa està posicionat. Un cop avaluada la posició, avalua si la posició del mapa ha arribat a un extrem, per saber si s'ha de moure el fons o el personatge.

```
void scrollingMapaHoritzontal(Mapa *m, Hero *p){
    if((*m).getPositionX()+(Pad.Held.Right - Pad.Held.Left) >= 0 and
    (*m).getPositionX()+(Pad.Held.Right - Pad.Held.Left) <= (*m).getLimitX()){

        if (Pad.Held.Right){
            if(canMoveRight(getTileXPlayer(m,p),getTileYPlayer(m,p)) || Pad.Held.Up || Pad.Held.Down){
                (*m).setPositionX((*m).getPositionX()+(Pad.Held.Right - Pad.Held.Left));
            }
        }

        if (Pad.Held.Left){
            if(canMoveLeft(getTileXPlayer(m,p),getTileYPlayer(m,p)) || Pad.Held.Up || Pad.Held.Down){
                (*m).setPositionX((*m).getPositionX()+(Pad.Held.Right - Pad.Held.Left));
            }
        }
        PA_EasyBgScrollXY(ScreenDown,(*m).getNFondo(),(*m).getPositionX(), (*m).getPositionY());
    }
}
```

```
void scrollingMapaVertical(Mapa *m, Hero *p){

    if((*m).getPositionY()+(Pad.Held.Down - Pad.Held.Up) >= 0 and
    (*m).getPositionY()+(Pad.Held.Down - Pad.Held.Up) <= (*m).getLimitY()){

        if (Pad.Held.Up){
            if(canMoveUp(getTileXPlayer(m,p),getTileYPlayer(m,p)) || Pad.Held.Left || Pad.Held.Right){
                (*m).setPositionY((*m).getPositionY()+(Pad.Held.Down - Pad.Held.Up));
            }
        }

        if (Pad.Held.Down){
            if(canMoveDown(getTileXPlayer(m,p),getTileYPlayer(m,p),p) || Pad.Held.Left || Pad.Held.Right){
                (*m).setPositionY((*m).getPositionY()+(Pad.Held.Down - Pad.Held.Up));
            }
        }
        PA_EasyBgScrollXY(ScreenDown,(*m).getNFondo(),(*m).getPositionX(), (*m).getPositionY());
    }
}
```

En el codi anterior podem veure els 2 mètodes encarregats de controlar i executar el codi per al moviment del mapa (tant en vertical com en horitzontal). El codi avalua que la següent posició del mapa no sigui un dels límits i en cas afirmatiu avalua si la següent posició on es mourà hi detecta una col·lisió. En cas de no detectar col·lisió s'actualitza el mapa amb la nova posició.

```
void scrollingPersonatgeHoritzontal(Mapa *m, Hero *p){

    s32 movX,movY;

    movY=(*p).getPositionY();
    movX=(*p).getPositionX()+(Pad.Held.Right - Pad.Held.Left);

    if(movX>=0 and movX<240){

        if (Pad.Held.Right){
            if(canMoveRight(getTileXPlayer(m,p),getTileYPlayer(m,p)) || Pad.Held.Up || Pad.Held.Down){
                (*p).setPositionX(movX);
            }
        }

        if (Pad.Held.Left){
            if(canMoveLeft(getTileXPlayer(m,p),getTileYPlayer(m,p)) || Pad.Held.Up || Pad.Held.Down){
                (*p).setPositionX(movX);
            }
        }
        PA_SetSpriteXY(ScreenDown, (*p).getNSprite(), (*p).getPositionX(), (*p).getPositionY());
    }
}
```

```
void scrollingPersonatgeVertical(Mapa *m, Hero *p){

    s32 movX,movY;

    movX=(*p).getPositionX();
    movY=(*p).getPositionY()+(Pad.Held.Down - Pad.Held.Up);

    if(movY>=0 and movY<160){
        if (Pad.Held.Up){
            if(canMoveUp(getTileXPlayer(m,p),getTileYPlayer(m,p)) || Pad.Held.Left || Pad.Held.Right){
                (*p).setPositionY(movY);
            }
        }

        if (Pad.Held.Down){
            if(canMoveDown(getTileXPlayer(m,p),getTileYPlayer(m,p),p) || Pad.Held.Left || Pad.Held.Right){
                (*p).setPositionY(movY);
            }
        }
        PA_SetSpriteXY(ScreenDown, (*p).getNSprite(), (*p).getPositionX(), (*p).getPositionY());
    }
}
```

En els mètodes anteriors s'avalua a quina posició respecte la pantalla es troba el personatge principal. Després s'avalua si es pot desplaçar a la següent posició sense detectar col·lisió. En cas afirmatiu s'actualitza la posició del personatge amb la nova posició.

9.3 Col·lisions

En aquest projecte s'han definit 2 tipus de col·lisions per al personatge principal: les col·lisions amb elements estàtics i les col·lisions amb elements dinàmics.

Les col·lisions amb elements estàtics requereixen d'una feina extra al haver de crear

el mapa de col·lisions associat i afegir a les col·lisions els elements estàtics del mapa seleccionat.

9.3.1 Col·lisions amb el mapa

Per a realitzar les col·lisions amb de l'escenari i els elements estàtics s'han fet servir 2 grups d'imatges. El primer grup d'imatges es correspon amb les imatges associades als elements estàtics del joc, és a dir, cofres, ítems i NPC. El segon grup es correspon amb la imatge del mapa i la imatge del seu mapa de col·lisions.

Un mapa de col·lisions és un mapa amb les mateixes mides que el mapa que volem afegir al joc i que està dividit en tiles. Cada tile té un color pla que equivaldria a un tipus de tile (blanc es pot travessar, Magenta no es pot travessar). A la Figura 39 es pot veure un exemple de mapa junt amb el mapa de col·lisions associat.

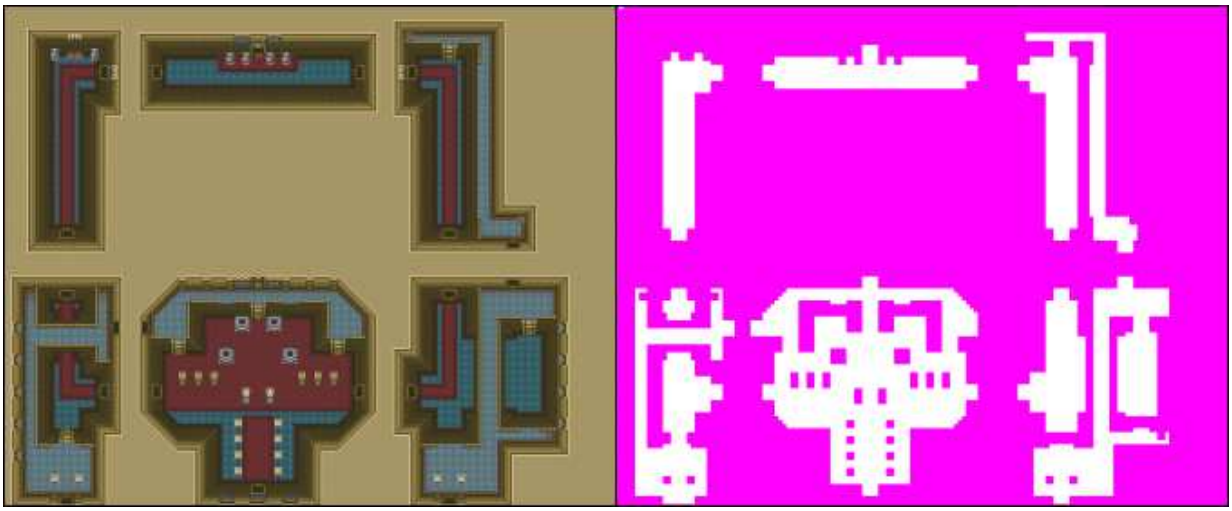


Figura 39: A l'esquerra un mapa i a la dreta el mapa de col·lisions associat.

El mapa de col·lisions es necessari a l'hora de fer la conversió de les imatges amb PAGfx, ja que al fer la conversió el programa genera un arxiu amb una matriu de tiles on a cada posició de la matriu s'hi guarda el color de la tile. L'únic requeriment del mapa de col·lisions que hi ha, és que a l'extrem superior esquerra s'han d'afegir tantes tiles de colors diferents com tiles diferents hi haurà en el mapa. L'ordre amb què posem els colors definirà quin valor es guardarà a la matriu (0,1,2,3 ...) seguint l'ordre. A la Figura 40 es pot veure un exemple de la relació color - valor del mapa de

col·lisions.

Figura 39: Extrem superior esquerra d'un mapa de col·lisions amb els valors associats per a cada color de tile.

Per detectar una col·lisió amb el personatge principal s'ha definit que les col·lisions es detectaran respecte la base de la imatge del personatge principal. A la Figura 40 es pot veure marcat amb negre l'esquema de col·lisions del personatge principal.

Figura 40: Zona de col·lisió del personatge marcada amb negre.

Per calcular si el personatge és pot desplaçar en una de les 4 direccions, un cop avaluat en els mètodes d'scrolling quina tecla del pad direccional s'ha polsat, s'avalua si a la següent posició no hi ha col·lisió amb un dels següents mètodes:

```
bool canMoveLeft(s32 tileX, s32 tileY){
    s32 col1, col2;
    bool res = false;

    col1 = getCollision(tileX-1,tileY+2);
    col2 = getCollision(tileX-1,tileY+3);

    res = (col1 != 0 & col2 !=0);
    return res;
}

bool canMoveRight(s32 tileX, s32 tileY){
    s32 col1, col2;
    bool res = false;

    col1 = getCollision(tileX+2,tileY+2);
    col2 = getCollision(tileX+2,tileY+3);

    res = (col1 != 0 & col2 !=0);
    return res;
}

bool canMoveUp(s32 tileX, s32 tileY){
    s32 col1, col2;
    bool res = false;

    col1 = getCollision(tileX,tileY+1);
    col2 = getCollision(tileX+1,tileY+1);

    res = (col1 != 0 & col2 !=0);
    return res;
}

bool canMoveDown(s32 tileX, s32 tileY, Hero *p){
    s32 col1, col2;
    bool res = false;

    col1 = getCollision(tileX,tileY+4);
    col2 = getCollision(tileX+1,tileY+4);

    res = (col1 != 0 & col2 !=0);

    return res;
}
```

Els mètodes anteriors retornen cert si la funció getCollision (s32 x, s32 y) retorna per a les 2 posicions un valor diferent a 0 (no hi ha col·lisió).

```
s32 getCollision(s32 tileX, s32 tileY){
    s32 num = tileY * tamanyXTiles + tileX;
    return HC_1F_COL_Map[num];
}
```

Aquesta funció retorna el valor que ocupa a la matriu la tile amb coordenades (tileX, tileY).

A la Figura 41 podem veure com el personatge principal col·lideix amb el mapa per la part superior esquerra de la base del personatge.

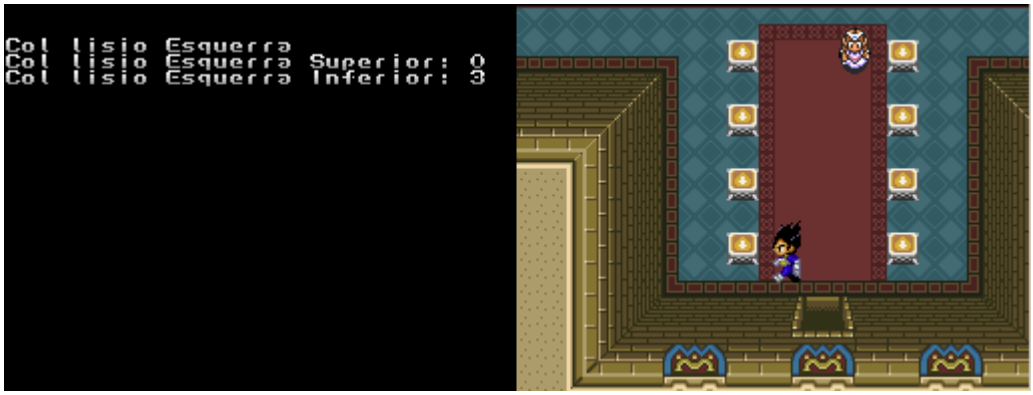


Figura 41: Exemple de col·lisió esquerra.

9.3.2 Col·lisions amb elements dinàmics

Les col·lisions que s'han definit en aquest projecte per a els elements dinàmics són col·lisions laterals. Això ve donat per l'algoritme de IA que s'ha definit per al moviment dels enemics.

```
bool Npc::collision(Hero *p){
    return (*p).getPositionMapX()+migPantallaX == getPositionX()-16 ||
    (*p).getPositionMapX()+migPantallaX == getPositionX()+16;
}
```

En el codi anterior podem veure com es fa el tractament d'avaluació de les col·lisions laterals. A la Figura 42 podem veure un exemple de col·lisió per l'esquerra, entre el personatge principal i un enemic.

Figura 42: A l'esquerra podem veure com els elements no col·lideixen. A la dreta un exemple de col·lisió.

9.4 Gestió d'events

Per a que un element de l'escenari pugui interactuar amb el personatge principal s'han de complir 2 condicions: la primera que la distància entre el personatge principal i l'element de l'escena estiguin a una distància igual o menor a una distància determinada; la segona si estem a una distància correcte que li provoquem un esdeveniment (mitjançant per exemple un input d'una tecla determinada).

```
bool Element::evaluate(Hero * h, u16 key){
    return (*ev).evaluate(h, key, this);
}
```

La funció anterior de la classe Element és l'encarregada d'avaluar si a un element determinat li estem provocant un esdeveniment. Aquesta funció crida a la següent funció de la classe Event:

```
bool Event::evaluate(Hero *h, u16 key_pressed, Element *e){
    return (*Con).evaluate(h,key_pressed,NULL) && (key ==0 || key == key_pressed);
}
```

La funció anterior és l'encarregada d'avaluar si la distància i la tecla que s'està polsant son adients per a provocar un esdeveniment. Aquesta funció crida a la següent funció de la classe isClose:

```
bool isClose::evaluate(Hero *p, u16 key, Element *e){

    return distance_2((*p).getPositionMapX(),(*p).getPositionMapY(),itPosX,itPosY) <= D_2;
}
```

La funció anterior avalua si la distància al quadrat entre els dos elements és menor o igual que la distància al quadrat predefinida per a poder interactuar amb l'element.

A la següent funció es mostra el càlcul de la distància al quadrat entre l'element i el personatge principal.

```
s32 isClose::distance_2(s32 perPosX, s32 perPosY, s32 itPosX, s32 itPosY){
    s32 x,y;
    x=(perPosX+128-itPosX);
    y=(perPosY+96-itPosY);
    return (x*x+y*y);
}
```

Un cop provoquem un esdeveniment a un element aquest executa la seva acció predefinida. En el següent codi es pot veure els diferents comportaments que s'han definit per als diferents elements de joc.

```
bool action::evaluate(Hero *p, u16 key, Element *e){
    bool b = false;

    switch(type){
        //0 -> Potion, 1 -> NPC, 2 -> Enemy, 3 -> Chest
        Npc *np;
        Chest *temp;
        s32 costat;
        case 0:
            (*p).setLife((*p).getLife()+life_Potion);
            b = true;
            break;
        case 1:
            PA_OutputText(ScreenUp,0,10,"T'estava esperant.");
            PA_OutputText(ScreenUp,0,11,"Entra i mou-te per el castell.");
            PA_OutputText(ScreenUp,5,20,"Prem X per continuar.");
            break;
        case 2:
            np = dynamic_cast<Npc*>(e);
            PA_OutputText(ScreenUp,0,2,"Vida Enemic: %d",(*np).getLife());
            costat=(*np).acostarEnemic(p);
            if((*np).collision(p)){
```

```
                if (costat ==1){
                    (*np).setPositionX((*np).getPositionX()-30);
                }
                else if(costat ==2){
                    (*np).setPositionX((*np).getPositionX()+30);
                }
                (*p).setLife((*p).getLife()-life_atac);
            }
            else if ((*np).isTarget(p) && Pad.Newpress.A){
                if (costat ==1){
                    (*np).setPositionX((*np).getPositionX()-30);
                }
                else if(costat ==2){
                    (*np).setPositionX((*np).getPositionX()+30);
                }
                (*np).restLife();
            }
            break;
        case 3:
            temp = dynamic_cast<Chest*>(e);
            (*temp).setNImage(chest_opened);
            (*temp).open();
            break;
    }
    return b;
}
```

A la Figura 43 podem veure un exemple on el personatge principal provoca un esdeveniment a un cofre. Aquest esdeveniment es produeix al estar a una distància al quadrat igual o inferior a 10 píxels i es polsa la tecla A del pad.

Figura 43: Exemple d'esdeveniment sobre un cofre.

9.5 Menú pausa

El menú pausa s'activa des de la pantalla de joc. En aquest menú el programa mostra un llistat de tots els objectes que té el personatge principal i permet gestionar l'ús d'aquests.

```
        if (Pad.Newpress.Start){
            //pausa (menu)
            bool continuar = false;
            while(!continuar){
                PA_OutputText(ScreenUp,5,5,"Inventari: ");
                PA_OutputText(ScreenUp,5,20,"Prem Select per sortir.");
                continuar = llistarInventari(Heroi);
                if(Pad.Newpress.Select){
                    continuar = true;
                }
            }
            PA_ClearTextBg(ScreenUp);
        }
```

En el fragment de codi anterior es pot veure com es crida al menú de pausa des del programa principal.

```
bool llistarInventari(Hero * p){
    bool continuar = false;
    if ((*p).itemsInBag()!=0){

        for(u16 i =0; i<(*p).itemsInBag();i++){
            PA_OutputText(ScreenUp,0,6+i,"Pocio %d punts de vida",life_Potion);
        }
        while (!continuar){
            PA_OutputText(ScreenUp,0,17,"Què vols fer amb el primer Item?");
            PA_OutputText(ScreenUp,1,18,"Utilitzar (prem X)");
            PA_OutputText(ScreenUp,1,19,"Descartar (prem Y)");
            if(Pad.Newpress.X){
                (*(*p).itemBag(0)).makeAction(p,getKeyPressed());
                (*p).dropItemBag(0);
                continuar = true;
            }
            else if (Pad.Newpress.Y){
                (*p).dropItemBag(0);
                continuar = true;
            }
            else if (Pad.Newpress.Select){
                continuar = true;
            }
        }

    }
    else{
        PA_OutputText(ScreenUp,0,6,"Inventari buit");
    }
    return continuar;
}
```

Com es pot veure en el codi anterior, es mostren per pantalla tots els ítems de l'inventari del personatge principal i permet fer una gestió senzilla dels elements.

10 Resultats

S'han realitzat tots els apartats planificats inicialment. A continuació es poden veure diferents captures del treball realitzat:

10.1 Captures

A la Figura 44 es pot veure la pantalla inicial un cop arrancada la demo.

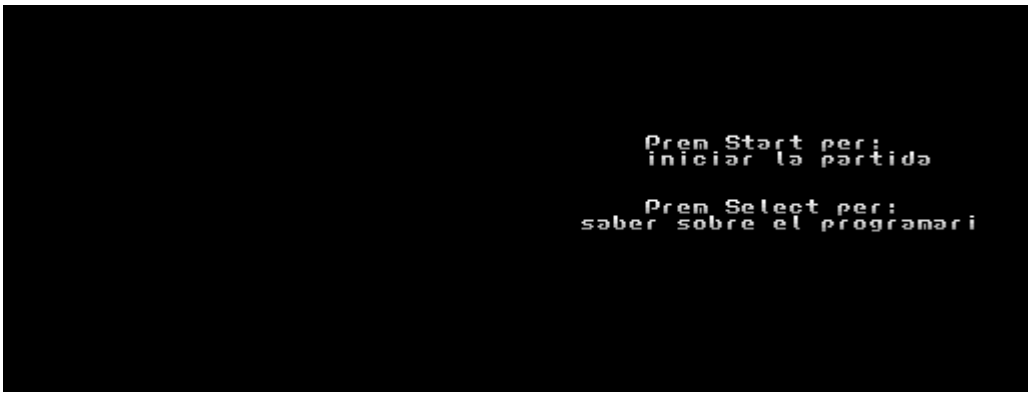


Figura 44: Screenshot – Pantalla Inicial.

A la Figura 45 es pot veure la pantalla "saber sobre el programari".

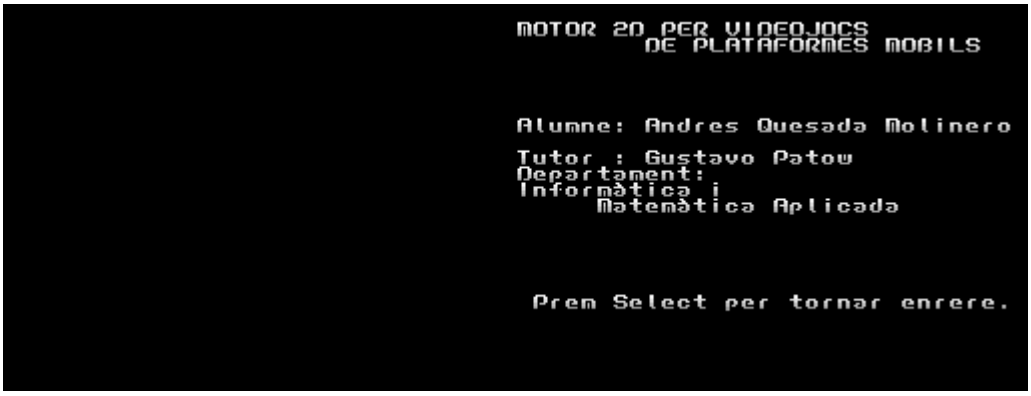


Figura 44: Screenshot – Pantalla Inicial.

A la Figura 46 es pot veure la pantalla un cop iniciada la demo.

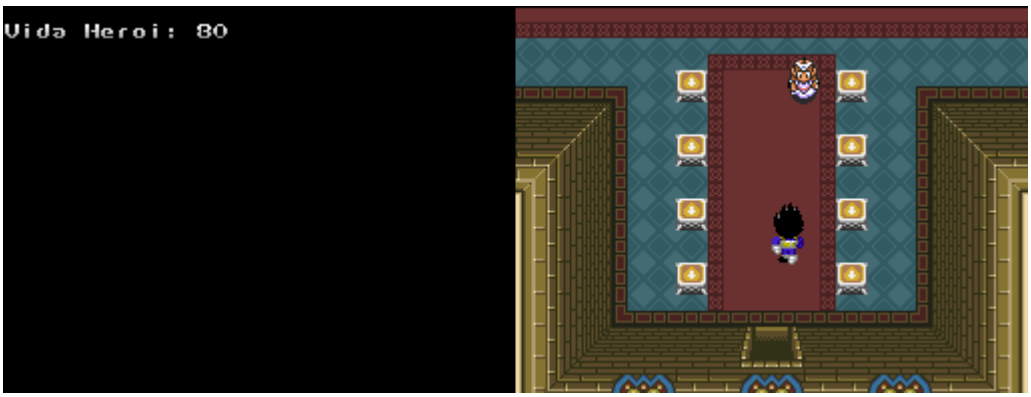


Figura 46: Screenshot – Inici de la demo.

A la Figura 47 podem veure com interactua el personatge principal amb un NPC.

Figura 47: Screenshot – Interacció entre heroi i NPC.

A la Figura 48 podem veure com interactua el personatge principal amb un ítem.

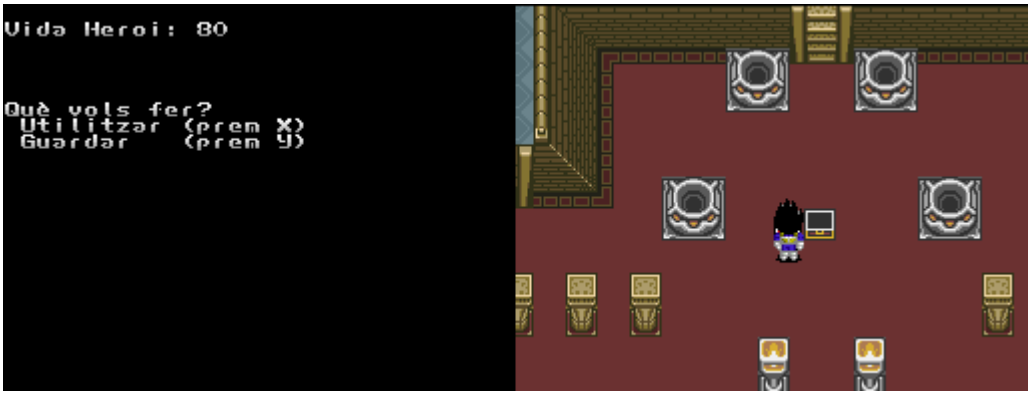


Figura 48: Screenshot – Interacció entre heroi i ítem.

A la Figura 49 podem veure com interactua el personatge principal amb un enemic.

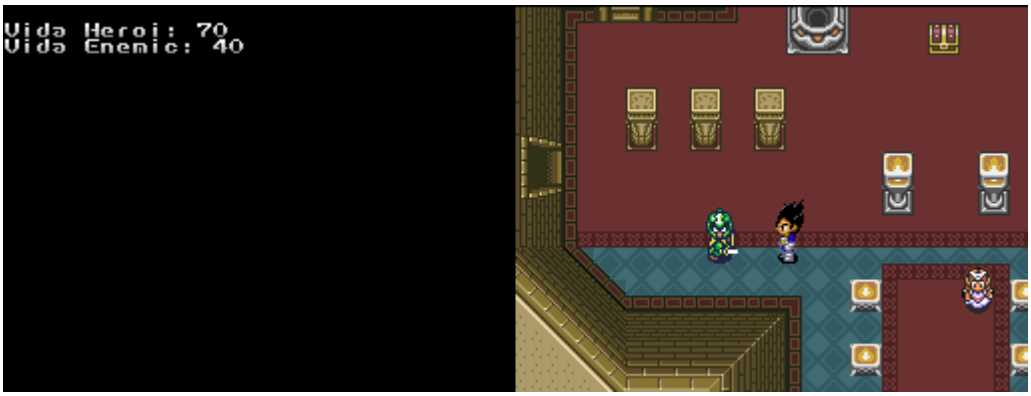


Figura 49: Screenshot – Combat entre heroi i enemic.

A la Figura 50 podem veure com es mostra el menú de pausa i es pot interactuar amb els ítems guardats a la motxilla de l'heroi.

Figura 50: Screenshot – Menú de gestió d'inventari.

10.1 Vídeo

A la següent adreça web es pot veure un vídeo Ingame de la demo del motor de joc 2D per a plataformes mòbils.

http://www.youtube.com/watch?v=fQcSsvDM1CI

11 Conclusions

En aquest apartat es definiran l'assoliment dels diferents objectius proposats en aquest projecte.

11.1 Temporalització

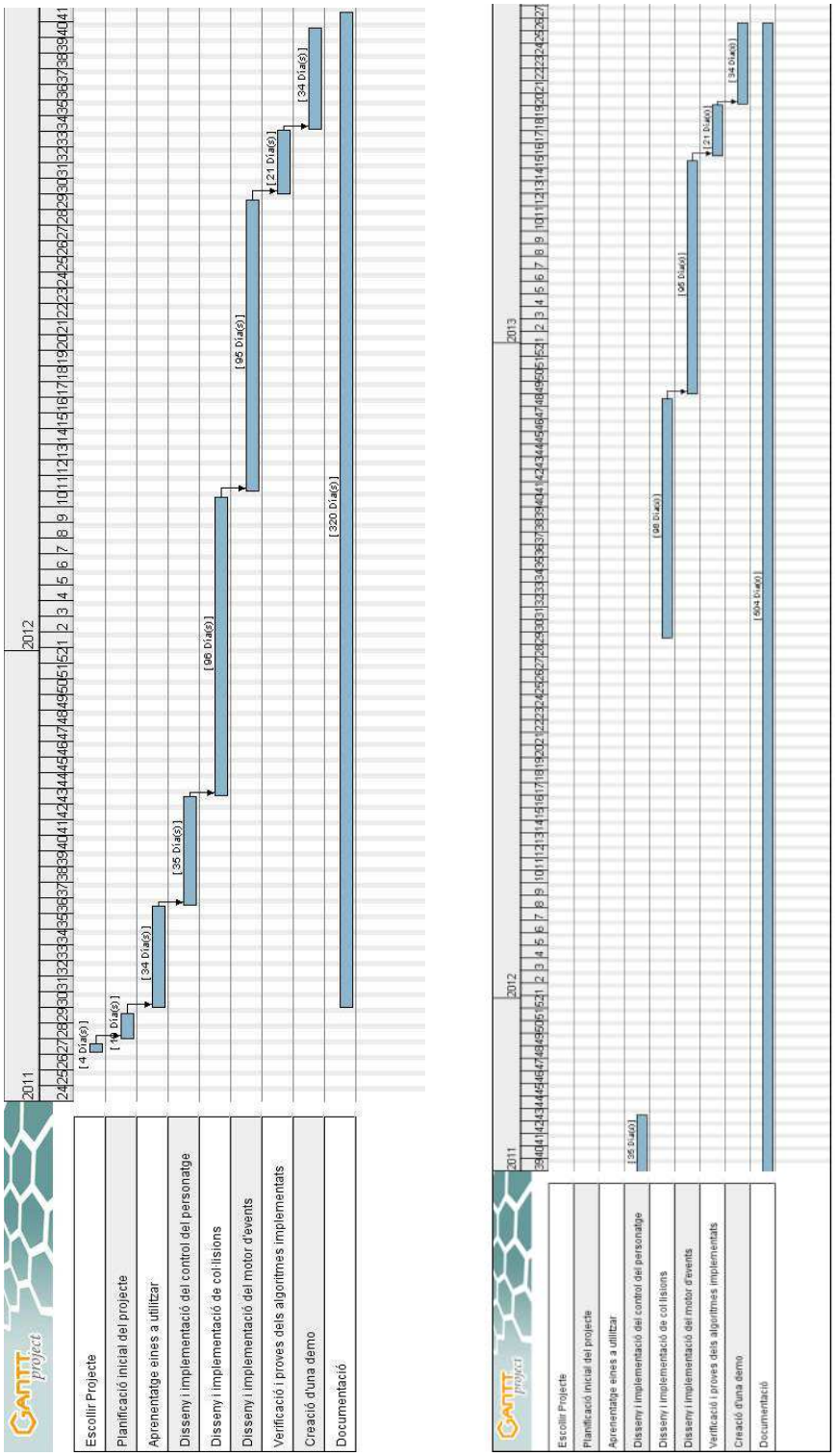


Figura 51: Diagrama de Gantt inicial (esquerra) i final (dreta).

Poques vegades la temporalització planificada inicialment en un projecte es correspon amb la que s'obté en realitat. A la Figura 51 podem veure la temporalització inicial i la que es correspon amb la realitat.

Per motius laborals es va haver de parar el desenvolupament del projecte durant uns 10 mesos i per això no ha estat possible seguir la temporalització inicial i el projecte.

11.2 Objectius

Tot i el problema de la temporalització es pot dir que en aquest projecte s'han assolit els següents objectius, al crear un motor de joc a partir d'unes eines bàsiques de mostreig de gràfics i interacció amb el hardware que:

- Permet executar un algorisme d'scrolling tant al personatge principal com al mapa.
- Permet l'ús de col·lisions tant amb elements estàtics com elements dinàmics.
- Permet la interacció del personatge principal amb diferents elements de la pantalla tant en temps real com amb elements guardats en inventari.
- S'ha programat un motor d'esdeveniments modular on es pot afegir / modificar comportaments i condicions.
- S'ha creat una llibreria robusta capaç de ser executada tant en un dispositiu de Nintendo (Nintendo DS), com en dispositius més nous de Nintendo (Nintendo 3DS) o dispositius de Sony (PSP Vita).

  En el cas de Nintendo 3DS es podria executar el binari de forma nativa.

  En el cas de PSP Vita s'hauria de compilar amb el compilador PAlib de PSP Vita i realitzar algun petit canvi a nivell d'ús de llibreries PAlib.

11.3 Conclusions

Durant l'elaboració d'aquest projecte s'ha arribat a les següents conclusions:

- És fonamental utilitzar les eines adequades i realitzar un bon anàlisi del que es vol realitzar i quins seran els requisits funcionals.

- La realització d’un motor de joc no és senzilla. Hi ha moltes feines associades que no tenen a veure amb la programació però que sense elles no es podria tirar endavant el desenvolupament, com la tria dels gràfics, comportaments d’elements ...
- L’objectiu personal de crear un motor de joc i crear una demo que demostrés l’assoliment dels objectius ha estat molt satisfactori. A més aquest motor de joc es pot completar afegint noves funcionalitats i/o amb els elements disponibles crear un petit videojoc.

12 Treball futur

Aquest projecte es pot ampliar i/o millorar amb les següents tasques proposades:

- Millora de la IA dels enemics. Es podria substituir l'algoritme actuar per A*.
- Afegir animació gràfica als enemics.
- Afegir un sistema de so, tant com música i efectes sonors.
- Millora del sistema actual de lluita amb enemics per un més atractiu visualment.
- Disseny de més tipus d'ítems amb noves funcionalitats.
- Afegir un sistema d'scrolling multinivell, és a dir, no treballar amb un mapa pla sinó treballar amb un mapa amb diferents alçades.
- Millorar la gestió d'ítems inventariats per un sistema amb més funcionalitats i més visual.

13 Bibliografia

En aquesta secció es llisten totes les fonts consultades per a la realització del projecte.

devkitPro. 2009. devkitPro.org. 30 juliol 2011. http://devkitpro.org/

ElOtroLado.net. 1999. New EOL, SL. 30 juliol 2011. http://www.elotrolado.net/

NDS Scenebeta.com. 2005. SB IT MEDIA, SL. 3 agost 2011. http://nds.scenebeta.com/tutorial/recopilaci-n-de-tutoriales-de-palib

GSA. 2002. GSArvhives.net. 20 agost 2011. http://www.gsarchives.net/

El blog de los videojuegos. 2009. Flavio Escribano. 15 desembre 2011. http://videojuegos.leer.es/

Comunidad RPG Maker. 2002. Comunitat RPG Maker. 15 desembre 2011. http://comunidad.rpgmaker.es

ZeldaShrine. 2003. FFshrine.org. 12 gener 2011. http://zs.ffshrine.org/link-to-the-past/sprites.php

Nintendo Software Development Support Group. 2013. Nintendo of America Inc. 20 juny 2013. http://www.warioworld.com/

GBADEV. 2001. Gbadev.org. 15 gener 2011. http://forum.gbadev.org/viewforum.php?f=18

TICBeat. TICbeat. 7 abril 2012. http://www.ticbeat.com/

La web del programador. 2000. 30 juliol 2011. http://www.lawebdelprogramador.com/

14 Manual d'instal·lació

Aquest manual d'instal·lació no va dirigit a un usuari final, sinó que va dirigit a un usuari programador. Per tant, el que s'explicarà en aquest apartat és el mètode d'instal·lació del Kit de desenvolupament per a NDSLite.

El primer que farem serà descarregar la última versió del devkitPro de la pàgina www.devkitpro.org. Un cop baixat executem l'instal·lador i seguim els passos d'instal·lació com es pot veure a la Figura 52. L'únic requisit per a poder desenvolupar per NDSLite és marcar la opció devkitARM.

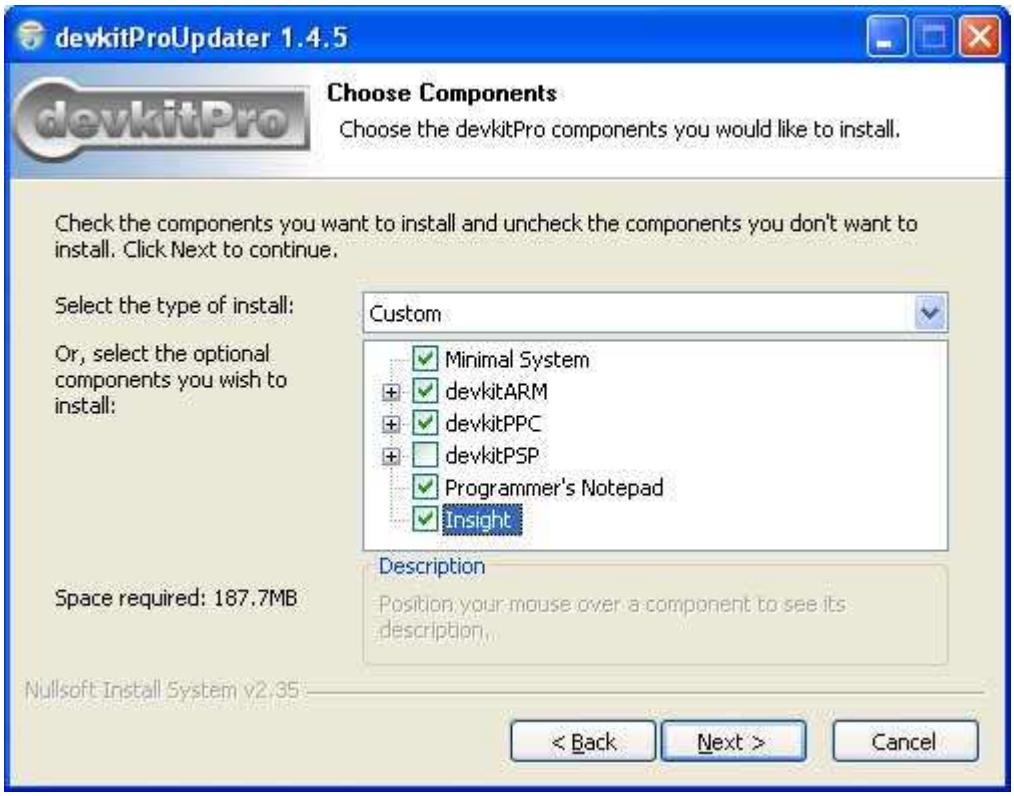


Figura 52: Programa d'instal·lació de devkitPro.

Un cop instal·lat l'entorn de desenvolupament i els compiladors, ens baixem la llibreria gràfica PAlib de la següent web:
http://sourceforge.net/projects/pands/files/Installer/PAlib%20070323/

A la Figura 53 podem veure el programa d'instal·lació de les llibreries PAlib.

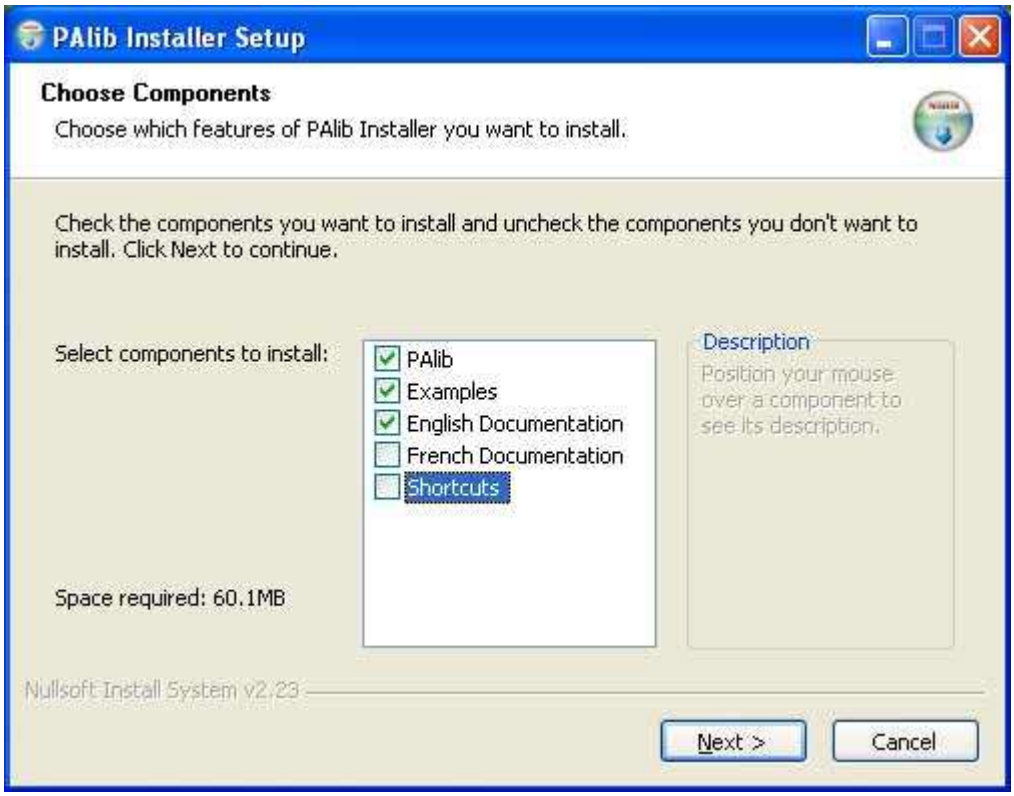


Figura 53: Programa d'instal·lació de PAlib.

Un cop instal·lada la llibreria PAlib, s'ha d'anar a la ruta on s'ha instal·lat devkitPro, i a dins al directori PAlibTemplate. En aquesta carpeta hi ha l'esquelet per a poder compilar el projecte. Copiem el contingut i el posem al directori de treball que fem servir habitualment. L'únic que ens interessa d'aquest esquelet són 2 coses: la carpeta source (on s'hi posa tot el codi font) i l'arxiu Build.bat (compila el projecte i genera el binari).

Dins la carpeta PAlib del directori on s'hagi instal·lat devkitPro, hi ha una carpeta tools amb diverses eines per a treballar amb la programació de NDS, entre elles es troba PAGfx, per a fer el tractament de les imatges.

Dins de la carpeta PAlib també s'hi troba la carpeta emulators, on hi ha un emulador per a poder executar el binari que en resulti de la compilació del projecte.