Intel·ligència artificial aplicada a un joc beat'em up en 2D

Memòria del projecte

d'Enginyeria Tècnica en

Informàtica de Sistemes

realitzat per

Carlos Adan López

i dirigit per

Maria del Mar López Hernandez

Escola d'Enginyeria

Sabadell, *juny* de *2011*

La sotasignat, ***Maria del Mar López Hernandez***,

professora de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria
ha estat realitzat sota la seva direcció per

Carlos Adan López

I per a que consti firma la present.
Sabadell, ***juny*** de ***2011***.

Signat: ***Maria del Mar López Hernandez***

FULL DE RESUM – PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte: Intel·ligència artificial aplicada a un joc beat'em up en 2D.	
Autor[a]: Carlos Adan López.	**Data:** ***juny de 2011***
Tutor[a]/s[es]: Maria del Mar López Hernandez.	
Titulació: Enginyeria Tècnica en informàtica de Sistemes.	

Paraules clau (mínim 3)

- Català: intel·ligència artificial, videojocs, BENNU, agents, concurrència.
- Castellà: inteligencia artificial, videojuegos, BENNU, agentes, concurrencia.
- Anglès: artificial inteligence, videogames, BENNU, agents, concurrence.

Resum del projecte (extensió màxima 100 paraules)

- Català:

Creació d'un joc del tipus Arcade beat'em up en 2D utilitzant escenaris amb una certa profunditat de moviment i dotant als personatges no jugadors i altres objectes d'Intel·ligència Artificial de manera que el seu comportament no sigui sempre lineal i aprofitant-ho per afegir nivells de dificultat.

- Castellà:

Creación de un juego del tipo Arcade beat'em up en 2D utilizando escenarios con una cierta profundidad de movimiento y dotando a los personajes no jugadores y otros objetos de Inteligencia Artificial de manera que su comportamiento no sea siempre lineal y aprovechándolo para añadir niveles de dificultad.

- Anglès:

Creating an Arcade game type Beat'em up in 2D using stages with some moving depth and providing the non playable characters and other objects with Artificial Inteligence so their behavior is not always linear and taking advantage with this to add some difficulty levels.

Índex

1. Introducció

1.1 Resum del projecte.

El projecte tracta bàsicament sobre l'estudi del funcionament d'un joc posant un interès especial en el seu aspecte intern més conegut com *engine*[1][RB13] o motor i en concret al desenvolupament de la part **d'intel·ligència artificial** (**I.A.**). Per fer-ho s'ha optat per dissenyar un motor senzill per un joc en 2D del tipus que es coneix com *beat'em up* [2][RB12].

L'*engine* tot i ser relativament senzill en comparació amb altres més professionals i actuals consta de tots els elements necessaris per poder fer funcionar el joc proporcionant-li una estètica típica dels jocs de la dècada dels 90. S'ha optat per aquest estil més aviat antic o aparentment desfasat tant per la seva simplicitat a l'hora de desenvolupar en comparació amb els moderns jocs en 3D com pel simple fet que no per ser més simples aparentment han de ser menys entretinguts. De fet encara avui hi ha una gran quantitat de gent que encara hi juga ja sigui per gust o per necessitat al no poder disposar de màquines més potents.

La **I.A.** de la que es parla a la memòria es refereix tant a la part de la interacció amb el jugador com al comportament dels enemics en funció d'unes determinades condicions tot i que cal distingir entre la **I.A.** que es definiria com intel·ligència **basada en el comportament del personatge** i la intel·ligència que fa referència a **altres comportaments com ara les col·lisions, estratègies i reaccions entre els agents**.

La intel·ligència dels personatges inclou tant enemics, protagonistes o fins i tot objectes. És a dir, tot el que es pot considerar un **agent intel·ligent**. Funciona mitjançant **FSM**[3][RB1] [RB7] [RB17] (*Finite State Machines*) que en el cas del protagonista es combina amb la interacció de l'usuari. A més a més la **concurrència** és un tema que està molt present al treball ja que a més de la sincronització dels diferents processos també es pot aplicar com a tècnica per desenvolupar el seu comportament.

Tota la resta d'intel·ligència engloba tant el sistema de col·lisions (reaccions degudes a algun contacte amb altres agents), interacció dels escenaris (si cal) i gestió dels mateixos així com de la dificultat, que s'adaptarà al perfil del jugador, i les modificacions que aquesta aplica ja que pot augmentar o disminuir la agressivitat, la quantitat d'enemics i el seu número de reaccions fent-los automàticament més o menys intel·ligents.

1.2 Motivacions del projecte.

La motivació principal ha estat la del plantejament d'un repte personal i veure fins a quin punt es podrien aprofitar els coneixements adquirits a la carrera aplicats a un tema tan obert i amb tantes possibilitats com el dels videojocs, tenint en compte que cada tipus de joc requereix unes tècniques d'implementació de **I.A.** i en general una forma diferent d'enfocar tot el seu desenvolupament.

---

[1] De forma genèrica un engine són un conjunt de rutines de programació que permeten el disseny, la creació i la representació d'un videojoc. La funcionalitat bàsica és la de proporcionar un motor de renderització (tractament d'imatges 3D) si cal, un motor físic o de col·lisions, so, scripts, animació, I.A. i altre components que puguin ser útils.

[2] Tipus de videojoc que consisteix en anar avançant pels nivells derrotant gran quantitat d'enemics amb o sense armes.

[3] Una Màquina d'estats finits és un autòmat amb sortides. Generalment en jocs (i en el projecte) s'utilitza el model de la màquina de Moore ja que les accions dels agents es fan al seu estat corresponent.

Per altra banda cal afegir que sempre s'ha jugat a aquests tipus de jocs (independentment del gènere) des de la època del gran èxit de les companyies Nintendo i SEGA i es podria dir que va ser precisament aquesta afició la que a la llarga va influir a realització d'aquests estudis. Això juntament amb el fet d'escoltar cada cop més que la majoria dels jocs moderns prioritzen altres aspectes en comptes de l'entreteniment van fer que la idea anés agafant més consistència i aprofitant la quantitat d'eines de que es disposa avui dia tant a nivell de programació com a nivell d'entorns de desenvolupament de jocs es va decidir fer aquest projecte.

El motiu pel qual es va triar un *beat'em up* va ser precisament perquè poden oferir un nivell d'intel·ligència a temps real tant elaborat com sigui necessari i sense ser necessàriament complicat a l'hora d'implementar.

1.3 Objectius del projecte.

L'objectiu principal del projecte és el de realitzar la part de l'*engine* d'un joc que fa referència a la **Intel·ligència artificial** tant a nivell d'interacció amb el jugador com a nivell de comportament dels diferents **agents** ja siguin enemics, objectes, escenaris o el protagonista.

Dins d'aquests comportaments s'inclouen les diferents característiques:

- Dificultat adaptativa a les necessitats del jugador. Fet que proporciona una sensació d'aprenentatge del propi joc.
- Enemics intel·ligents. De manera que tenen un comportament que pot ser independent o no al dels agents companys.
- Escenaris dinàmics en funció de la dificultat. A partir de determinades dificultats poden sorgir esdeveniments que amb una dificultat menor no passarien.
- Interacció usuari-protagonista tant a nivell de jugabilitat per teclat com a les dificultats de l'escenari.
- Sistema de col·lisions i tractament d'aquestes.

Com és d'esperar per poder comprovar el funcionament de l'*engine* cal realitzar un joc on s'incloguin aquests objectius de manera que el resultat final sigui un joc de demostració (demo jugable).

1.4 Resum dels continguts de la memòria.

A continuació es pot trobar l'estat de l'art, on es fa una explicació de la evolució dels videojocs en els últims anys i com aquesta evolució ha afectat la forma de programar-los i com els propis personatges van convertint-se en agents cada com més intel·ligents.

El següent capítol és el dedicat a l'estudi de viabilitat i s'inclou un anàlisi de les possibilitats d'aconseguir els objectius. A més, s'especifiquen els recursos que s'utilitzaran tant a nivell de programació com a nivell de gràfics. Per acabar l'apartat es podrà trobar la planificació.

Al quart es fa un anàlisi del projecte en el que s'especifiquen els requeriments tant funcionals com no funcionals i les restriccions del disseny que hi puguin haver.

Després es parla del disseny i es descriuen les diferents parts del joc com ara la interfície del joc, els escenaris i els aspectes relacionats amb la **I.A.** que s'ha implementat.

Dins del capítol de la implementació es descriuen els diferents processos, els estats que puguin tenir i les funcions més importants. També s'explica el funcionament dels escenaris.

El capítol de proves descriu la sèrie de tests i proves que s'han anat duent a terme al llarg de tot el desenvolupament.

A continuació es poden trobar les conclusions del projecte amb els resultats obtinguts.

El següent capítol parla de les línies de millora i ampliacions que pot tenir el projecte.

Després s'inclou la Bibliografia on es poden trobar les fonts d'informació consultades.

Finalment s'ha afegit un Annex a on es poden consultar explicacions o ampliacions d'alguns continguts de la memòria que no s'han explicat dins dels capítols anteriors.

2. Estat de l'art

Degut a la enorme evolució de la industria dels videojocs tant en els aspectes de hardware com de software no seria erroni pensar que tot i ser dos camins ben diferents alhora estan ben lligats. Per poder entendre-ho amb una mica més de claredat resulta útil fer un breu repàs històric en aquests dos aspectes.

Pel que fa a la **I.A.** al començament els jocs (entre els 60 i els 70) ni tan sols feien servir el que es podria dir una **I.A.** real (normalment aprofitaven un segon jugador o patrons molt simples) i més endavant van afegir millores. Simplement utilitzaven una sèrie de patrons (més complexes que els anomenats anteriorment) que definien el seu comportament, cosa que els feia previsibles i per solucionar-ho feien que el joc s'avancés als moviments del jugador aprofitant la informació de les comandes polsades. Cap als anys 90, ja amb màquines més potents i amb les tècniques d'intel·ligència més evolucionades van incorporar les **FSM** (Màquines d'estat Finits), que encara avui es fan servir. Actualment les tècniques més senzilles s'estan deixant de banda gradualment aprofitant el millor rendiment de les CPUs per deixar pas a unes rutines d'intel·ligència més sofisticades. Bàsicament s'inclouen gradualment les tècniques de programació de la **I.A.** enlloc dels patrons preestablerts. A la taula 2.1 es resumeix breument el *timeline* de la **I.A.** a l'àmbit dels jocs fins al 2001[RB1].

Taula 2.1: *Timeline* de la I.A. dels jocs fins al 2001.

No I.A. Real	Primer joc per computador ("Space War!") Requeria 2 jugadors	1962
	Surt a la venda el "Pong". Gran interès per a les masses.	1971
Patrons	Comencen a aparèixer els enemics. El jugador havia de disparar a un objectiu amb patrons de moviment.("Pursuit" i "Qwak").	1974
	Surt a la venda el "Gun Fight" amb un microprocessador, cosa que permetia més elements aleatoris i més computació.	1975
	Primer dels jocs considerats clàssics ("Space Invaders"). Enemics amb patrons, puntuacions, controls senzills, nivells i dificultat incremental.	1978
	"Galaxian" utilitza la fórmula d'"Space Invaders" però millorant la complexitat de moviment.	1979
	"Pac-Man" surt a la venda amb un moviment per patrons dels fantasmes però cadascun es comporta de forma independent.	1980
	Un microprocessador derrota a un mestre dels escacs per primer cop.	1983
	"Karate Champ" surt a la venda en la versió d'un jugador. És un dels primers jocs de lluita 1 vs 1 amb una I.A. molt simple. Més endavant va sortir la versió per 2 jugadors.	1984
FSMs	Primer joc d'estratègia en temps real("Herzog Zwei"). Primera incorporació del *pathfinding*[4][RB2] tot i que bastant erroni.	1990
	Surt a la venda el famós "Doom" per PC, oficialment inaugurant la era dels *Shooters en primera persona*[5][RB1].	1993
Altres Tècniques	"BattleCruiser:3000AD" es publica fent in primer intent de fer servir xarxes neuronals[6][RB2] a un joc comercial.	1996
	El supercomputador "Deep Blue" derrota al campió mundial d'escacs Gary Kaspàrov per primer cop (Finalment Kaspàrov va acabar derrotant la màquina amb un total de 4-2).	1997
	Surt a la venda "Half Life" i la seva I.A. és considerada com la millor fins a la època i encara avui dia es considera de les millors.	1998
	Black & White conté criatures amb autoaprenentatge per reincidència i per observacions.	2001

[4] Algorismes cercadors de camins per anar d'un punt 'A' a un punt 'B'.

[5] Jocs amb el principal objectiu d'avançar pels nivells eliminant els enemics amb diferents armes normalment de foc i amb una perspectiva de primera persona respecte al jugador

[6] Paradigma d'aprenentatge y processament automàtic inspirat en el funcionament del sistema nerviós dels animals. Es tracta d'un sistema d'interconnexió de neurones dins d'una xarxa que produeixen un estímul de sortida.

Si mirem breument els aspectes tècnics n'hi ha prou amb fixar-se en que antigament amb els processadors de 8 bits únicament es podia destinar entre un 1% i un 2% a la **I.A.** mentre que avui dia, amb unes màquines que disposen de bons processadors gràfics i bones CPUs es pot destinar entre un 10% i un 35% o fins i tot més. De manera que els agents podent tenir reaccions més intel·ligents simplement pel fet que els dispositius per reproduir els jocs no només antigament eren bastant més lents si no que a més tenien menys memòria i per tant no es podien executar tantes instruccions a la vegada sense ralentitzar el funcionament del joc. A més la incorporació progressiva als entorns en 3D han permès una major quantitat de moviments fins al punt de poder interactuar directament amb els escenaris d'una manera cada cop més realista.

Pel que fa al desenvolupament de videojocs és clar que existeixen grans empreses que produeixen jocs comercials amb resultats que van des de la simplicitat més rudimentària fins al que podrien ser obres d'art però des de fa uns anys que es va tornar a parlar d'un gènere que durant els 90 (degut precisament a que pràcticament tota la industria dels jocs es basava en les consoles i màquines recreatives): els anomenats jocs *indie* (independents). Aquests es caracteritzen principalment en ser jocs de baix pressupost (o directament sense) però que no per aquest fet han d'estar necessàriament destinats al fracàs. De fet no son pocs els que han venut (i amb èxit) aquests jocs. L'exemple més clar és el de "Minecraft" (http://www.minecraft.net) que ha aconseguit fer un gran negoci. Òbviament es tracta d'un cas excepcional però hi ha casos no tant destacats que també han vist el seu èxit tot i ser gratuïts com ara el *remake* del "Streets of Rage" fet pel grup de "Bombergames" (http://www.bombergames.net/sorr_project/) o jocs increïblement senzills però adictius com el "Syobon Action" (o Cat Mario), que simplement es una paròdia del famós Super Mario de Nintendo però amb un gat i fins fa poc tenia la fama de ser el joc més difícil fins a un nivell que el propi repte que suposava arribar al final era el seu punt fort. Això és possible pel fet que hi ha més facilitats. Per una banda els ordinadors han anat adquirint un paper important també per als jugadors amb les seves possibilitats i millor rendiment i per l'altra els creadors de les consoles modernes més importants (Wii, PSP, Xbox360, PS3, Nintendo DS...) entre altres han publicat les seves API's[7][RB19] que per un petit cost anual generalment permeten desenvolupar jocs per a aquestes màquines i també algunes d'aquestes permeten les aplicacions *homebrew*[8][RB20].

Pel que fa al desenvolupament es poden trobar *engines* gratuïts o de pagament[RB18] que faciliten en gran mesura aquesta tasca. Alguns dels més coneguts són *Ogre 3D* (http://ogre3d.org) i el ben conegut *Unreal Engine* (http://www.unrealengine.com) utilitzat tant per els propis jocs "Unreal" com per altres jocs d'altres companyies. Per altra banda també hi ha editors que permeten fer jocs sense necessitat de programar pràcticament.

[7] Interfície de programació d'aplicacions. Es tracta del conjunt de funcions, procediments y/o mètodes que ofereixen una biblioteca que es pot utilitzar per altres aplicacions.

[8] Software no oficial i usualment més "casolà" creat per programadors per a varies plataformes.

3. Estudi de viabilitat

3.1 Situació actual.

3.1.1 Descripció de la situació actual.

Tal com s'indica a la introducció el projecte consisteix en desenvolupar la **intel·ligència artificial** d'un videojoc que tot i la seva aparent simplicitat a nivell estètic permet obtenir un marge de reaccions a temps real per part dels agents considerablement ampli. Per fer-ho s'ha hagut de crear el seu *engine* ja que tot videojoc necessita el seu per a poder funcionar correctament, independentment del seu nivell d'elaboració. Cal tenir en compte que alguns estan prefabricats i faciliten aquesta tasca. És el cas de molts dels jocs moderns que disposen d'un motor gràfic més elaborat o d'empreses que reutilitzen els seus propis *engines* o els posen a disposició d'altres empreses o simples usuaris.

Últimament hi ha una tendència cada cop major d'aplicar diferents tècniques de disseny de la intel·ligència artificial combinades amb alguns dels mètodes que es solien fer servir antigament com és el cas per exemple de l'ús de **FSM**. Això és gracies a la major capacitat de còmput dels processadors a diferencia els jocs més antics.

3.1.2 Lògica del sistema.

La figura 3.1 mostra a nivell lògic el funcionament del programa. Quan el jugador fa algun moviment amb el personatge *l'engine* s'encarrega d'executar aquest moviment i si hi ha alguna reacció per part dels agents (per exemple: enemic atacat o un canvi de posició) i sempre mostra els resultats per pantalla.

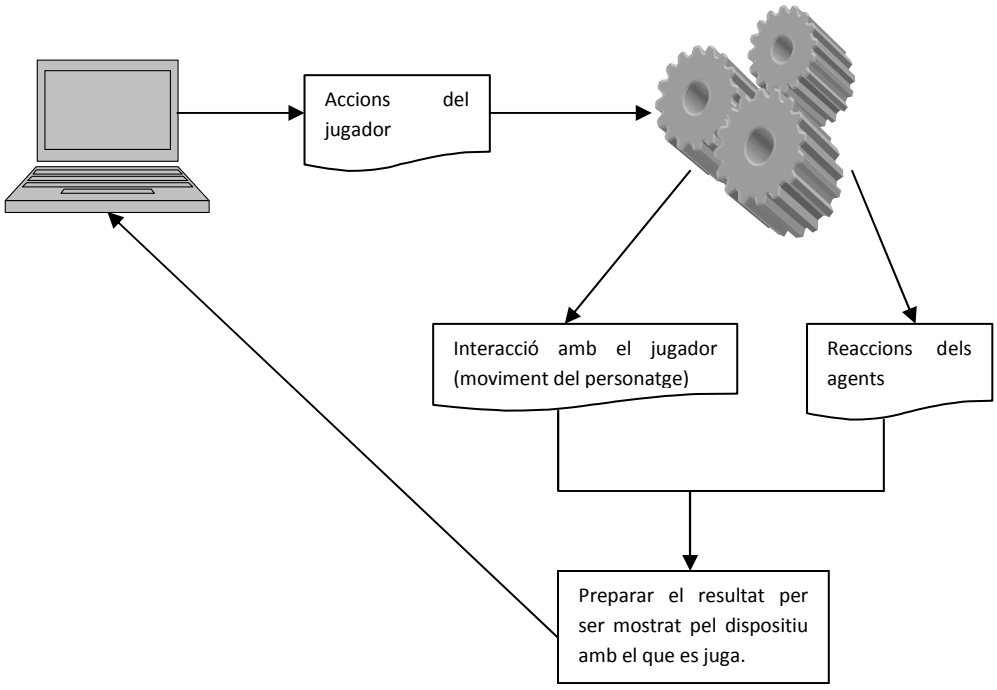


Figura 3.1: Lògica del sistema.

3.1.3 Descripció Física del sistema.

El joc està pensat per a poder funcionar amb pocs requisits.

PC:

Memòria RAM: 128 MB.

Processador: Pentium 4 2.8 Ghz.

Disc Dur: 42 MB.

Lector cd, DVD o USB.

Sistema Operatiu: MS Windows XP o superior/Linux.

Consoles:

GP2X Wiz o GP2X Canoo (Consoles preparades per fer funcionar aplicacions *homebrew* ja que tenen un Sistema Operatiu basat en Linux).

3.1.4 Diagnòstic del problema.

Un dels principals problemes és que fer un videojoc és un procés complex que normalment requereix de tot un equip per a la seva realització (analistes, dissenyadors gràfics, programadors, tècnics en so, testers...) Com a conseqüència de no disposar de la majoria dels recursos s'ha optat per una estètica més simple i així poder prioritzar la seva programació.

Per part dels aspectes tècnics el desenvolupament de la **I.A.** és una tasca complexa que ha de tractar tots els comportaments que calguin sense afectar a la jugabilitat ja que tot i que ha de suposar un repte la dificultat no ha de ser deguda a un mal funcionament o a uns enemics massa intel·ligents. També s'ha de tenir en compte el temps de resposta ja que al tractar-se d'un joc en temps real no és una bona idea fer servir algorismes massa complexos per a cadascun dels agents ja que al haver-hi una gran quantitat (dependrà de l'habilitat del jugador però podrien acumular-se fins a 15 aproximadament al mateix temps) podrien retardar el temps de resposta. Finalment i com és d'esperar el tema de la **concurrència**[9][RB4] [RB5] està molt present en el projecte ja que hi ha varis processos executant-se a la vegada i s'ha d'evitar que facin comportaments no desitjats, que puguin alterar el funcionament bàsic o puguin provocar errors.

3.2 Normatives i legislació.

1.- Normativa de projectes de final de carrera.

[9] En termes de programació es tracta de la sincronització entre els diferents processos per evitar accions no desitjades.

3.3 Anàlisi de requisits.

3.3.1 Requisits funcionals.

1.- Control dels moviments del personatge introduïts pel jugador (desplaçaments, salts i atacar).

2.- Control de l'*scroll*[10][RB8] [RB9] (mantenint el personatge sempre en pantalla).

3.- Gestió de la divisió dels escenaris en diverses parts.

4.- Gestionar els límits parcials, inicial i final dels escenaris.

5.- Evitar que el jugador travessi els límits sense complir el requisit previ (eliminar els enemics que quedin dins del límit).

6.- Gestionar les vides.

7.- Controlar la salut del personatge principal i que es mostri en tot moment.

8.- Controlar que si no li queden vides s'acaba el joc.

9.- Gestió de la salut dels enemics.

10.- Mostrar canvis físics als enemics que els hi quedi poca vida.

11.- Gestionar el temps restant per superar un dels límits (si es compleix el requisit per avançar reiniciar el temporitzador).

12.- Sincronització entre els diferents processos per evitar efectes indesitjats (concurrència).

13.- Gestionar la presa de decisions dels enemics.

14.- Gestionar la quantitat de reaccions possibles dels agents (enemics o objectes) segons la dificultat.

15.- Gestionar la puntuació durant i al final del nivell.

16.- Gestionar els canvis de dificultat.

17.- Controlar les dureses (un agent no pot caminar per les parets ni travessar-les).

18.- Gestió de les col·lisions i la física del joc (reaccions al contacte en cas de ser necessari).

19.- Gestió dels escenaris.

3.3.2 Requisits no funcionals.

1.- Els recursos (escenaris i *sprites*) han de ser de creació pròpia.

2.- La interfície ha d'informar de tot el que calgui sense incomodar al jugador.

[10] A un joc en 2D l'scroll fa referència al desplaçament de la pantalla.

3.- Ha de consumir pocs recursos.

3.3.3 Restriccions del sistema.

1.- L'aplicació s'ha de desenvolupar en entorn Windows o Linux.

2.- Les llibreries que no s'incloguin al sistema operatiu s'han d'incloure al producte final.

3.- El treball ha d'estar finalitzat abans del 28 de juny de 2011.

3.4 Alternatives i selecció de la solució.

Hi ha una gran quantitat de maneres de desenvolupar un joc. Algunes de les que més s'acosten al perfil dels requisits indicats a l'apartat anterior a són les que s'inclouen a continuació. Si no s'indica el contrari a les alternatives es consideren software gratuït.

Per mitjà d'*engines* prefabricats:

Allegro library: Bàsicament es tracta d'un conjunt de llibreries pel desenvolupament de videojocs i està preparat per funcionar amb C o C++ per defecte. Suporta tant gràfics en 2D com en 3D. Compatible amb DOS, Microsoft Windows, BeOS, Mac OS X i sistemes basats en Unix.

Lightweight Java Game Library: És un conjunt de llibreries de Java pel desenvolupament de jocs. És *Open Source*.

GameStudio: Es tracta d'un *engine* complet orientat a 2D i a 3D. Incorpora diferents nivells de programació (Principiant, Avançat i Professional). Utilitza un llenguatge similar a C. Incorpora una versió gratuïta (de prova) i altres versions de pagament.

Games Factory 2: Un software creador de jocs amb el qual gairebé no cal programar. És simple i senzill però no permet un control total de les situacions més complexes. Funciona amb Windows i la versió complerta més senzilla costa uns 99€.

Programant amb llenguatges directament:

C i C++: Podrien considerar-se els llenguatges de programació per excel·lència. Proporcionen un control de tot el que es vulgui fer tant a alt nivell com a baix nivell. La majoria de les llibreries estan implementades amb aquests llenguatges. Al ser compilats ofereixen un bon rendiment i gairebé tots els sistemes (si no tots) permeten compilar-los i executar el software un cop compilat amb el mateix sistema operatiu. C++ és la versió de C orientat a objectes. El principal problema es que si no s'està familiaritzat desenvolupar un software complex pot arribar a ser un procés costós.

Java: Llenguatge de programació orientat a objectes. És multiplataforma i requereix una màquina virtual de Java per poder funcionar (avui dia la majoria si no tots els sistemes operatius en porten una). El principal problema és que al requerir d'una màquina virtual el seu rendiment pot no ser tant òptim.

Lua: Es tracta d'un llenguatge orientat als scripts[11][RB21], és a dir, que es sol combinar amb altres llenguatges per mitjà d'scripts dins del codi. És senzill d'utilitzar i proporciona un rendiment molt bo alhora de tractar problemes que amb altres llenguatges podrien ser més complicats.

[11] Fragments de codi generalment en un llenguatge de programació diferent al que s'està fent servir per a la resta del programa.

BennuGD: També conegut com Bennu[RB9]. Es tracta d'un llenguatge de programació bastant recent i en contínua expansió per part de la comunitat orientat principalment per a la creació de jocs en 2D i en 3D. Les seves principals característiques són que està orientat a processos i que és de bastant alt nivell de manera que automatitza bastant la gestió dels recursos utilitzats i simplifica aspectes com el posicionament dels elements i les col·lisions. A més incorpora totes les llibreries necessàries (i se'n poden afegir d'opcionals). És multiplataforma (requereix una màquina virtual que sol venir inclosa amb el propi joc), tot i que el compilador permet fer un executable que inclogui la màquina virtual. Ofereix ports directes entre Windows, Linux i la consola GP2 Wiz entre altres i a més suporta altres sistemes com Mac OS i consoles com la Nintendo Wii, Xbox, PSP entre d'altres. De fet la llista de compatibilitats també es va ampliant. Estava distribuït sota la llicencia GNU GPL fins fa poc i recentment està sota llicencia Zlib.

ActionScript: Llenguatge orientat a objectes propi de Adobe Flash[12][RB22] i Flex[13][RB23] fet servir tant per animacions com per fer jocs relativament senzills (tot i que també n'hi ha algun joc bastant elaborat) principalment en 2D. Requereix una màquina virtual de Flash compatible amb la versió de l'ActionScript utilitzada. Tot i que les versions anteriors eren compatibles amb la PSP la versió 3.0 no ho és. El cost del software d'Adobe Flash (Si no és de prova) pot arribar als 825€ si es tracta d'un particular o 150€ en cas d'estudiants.

Tractament d'imatges:

Paint: Software per crear i editar imatges senzill incorporat als sistemes operatius de Windows per PC.

The GIMP: Software per la creació i edició d'imatges des del nivell més simple fins a un nivell professional amb multitud de característiques i ampliable amb extensions. Un altre avantatge que pot tenir respecte a altres editors d'imatges és que es pot descarregar gratuïtament tant per Linux com per Windows.

Paint.Net: Programa per editar imatges més complet que Paint i amb més possibilitats. Es pot descarregar gratuïtament per a Windows.

Adobe Photoshop: Software de gama professional per a crear i editar imatges que incorpora les mateixes característiques que GIMP (potser fins i tot més). Funciona amb Windows i Mac OS X però té un cost d'uns 169€ per a estudiants o 689€ per a particulars (o una versió de prova gratuïta).

Edició FPGs (contenidors de gràfics).

FPGEdit: Editor de FPGs que permet afegir i gestionar imatges en tres profunditats de color (8, 16 i 32 bits)

Smart FPG Editor: Similar a l'anterior editor però només funciona per Windows.

Edició de Fonts:

FNTMake: Programa que permet editar fonts de forma senzilla amb la possibilitat d'incloure una imatge com si fos el color de la font.

[12] Aplicació de creació i manipulació de gràfics vectorials(imatge digital formada a partir d'objectes geomètrics independents) amb possibilitats d'insertar codi mitjançant Action Script. Actualment està comercialitzat per Adobe Systems Incorporated.

[13] Conjunt de tecnologies creades per donar suport al desenvolupament d'aplicacions per internet basades en la plataforma Flash.

Solució triada:

Finalment s'ha decidit fer la implementació en *BennuGD* per diferents motius. Per una banda si es triava un *engine* prefabricat molt possiblement s'hauria d'haver combinat amb algun altre llenguatge (C, C++ o Java) si s'optava per alguna opció gratuïta i Game Factory no es podria haver utilitzat ja que no s'assegura la possibilitat de poder implementar la intel·ligència al nivell desitjat. Per altra banda **Bennu** proporciona les eines necessàries amb l'únic inconvenient (que tenen totes les opcions basades en el llenguatge) és que cal altre software per crear els recursos i al estar orientat als processos fa que la gestió entre els diferents elements sigui relativament senzilla (poden tractar-se com a processos independentment entre ells), a més del fet que ell mateix s'encarrega dels aspectes més interns com ara la gestió del DirectX[14] (o OpenGL[15]) i la gestió de la memòria. A més simplifica els aspectes més importants a l'hora del desenvolupament de jocs (col·lisions, càlcul de posicions i dureses dels elements). A més el fet de ser multiplataforma i d'estar en contínua evolució tant en funcionalitats com en termes de compatibilitat fa que sigui una opció adequada.

Per a la edició dels *sprites* s'ha optat per utilitzar **The GIMP** ja que permet el tractament ràpid de les imatges a més d'edició per capes i un control total dels canals de color (RGB i Alpha en imatges de 32 bits).

Per la creació i edició dels FPG s'ha optat per **FPG Edit** i per a la creació de les fonts **FNTMake** per la seva senzillesa.

3.5 Planificació del Projecte.

3.5.1 Tasques del projecte.

Anàlisi i Disseny de l'aplicació (temps estimat fins al 10/11/2010).

Disseny de la I.A.: Disseny dels diagrames de les màquines d'estats finits.

Disseny dels *sprites* i dels escenaris: Especificació del format de la imatge, profunditat de colors i qualitat de les imatges que s'utilitzaran.

Disseny de l'aplicació: Disseny de la interfície del joc i del funcionament intern (temporitzador, esdeveniments i gestió dels escenaris).

Disseny de les proves de Test: Disseny de les proves de funcionament per comprovar errors i el correcte funcionament de la I.A.

Desenvolupament de l'aplicació (temps estimat fins al 28/4/2011):

Creació dels sprites: Creació de les animacions i altres gràfics que s'utilitzaran i afegir-los als respectius fitxers .fpg.

Creació dels escenaris: Creació de les imatges que fan referència als escenaris (fons) amb els seus respectius mapes de dureses (plantilles en 8 bits que permeten indicar si és transitable o no).

Funcionalitats de l'aplicació: Programar el joc.

Proves i Tests finals (temps estimat fins al 9/5/2011).

[14] Col·lecció d'APIs desenvolupades per facilitar les tasques relacionades amb la multimèdia per a Microsoft Windows.

[15] Plataforma estàndard que defineix una API multillenguatge i multiplataforma per escriure aplicacions que produeixin gràfics 2D i 3D.

3.5.2 Planificació del projecte.

La figura 3.2 mostra la planificació temporal inicial del projecte. Cal dir que la generació de la memòria va començar una setmana abans degut a que el desenvolupament va acabar abans del que s'esperava.

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Asignación del proyecto	1 hora?	mié 29/09/10	mié 29/09/10
2	**Planificación**	**3,13 días**	**mié 29/09/10**	**lun 04/10/10**
3	Estudio viabilidad	3 días	mié 29/09/10	lun 04/10/10
4	Aprobación Estudio	1 hora	lun 04/10/10	lun 04/10/10
5	**Análisis de la aplicación**	**1 día**	**lun 04/10/10**	**mar 05/10/10**
6	Análisis de Requisitos (Casos de uso	1 hora	lun 04/10/10	lun 04/10/10
7	Requisitos de la I.A	1 hora	lun 04/10/10	lun 04/10/10
8	Documentación de la I.A.	2 horas	lun 04/10/10	lun 04/10/10
9	Análisis recursos multimedia	1 hora	lun 04/10/10	lun 04/10/10
10	Documentación recursos multimedia	2 horas	lun 04/10/10	mar 05/10/10
11	Aprobación Análisis	1 hora	mar 05/10/10	mar 05/10/10
12	**Diseño de la aplicación**	**26,5 días**	**mar 05/10/10**	**mié 10/11/10**
13	Diseño de la I.A	3 sem.	mar 05/10/10	mar 26/10/10
14	Diseño de Sprites y Mapas	1 sem	mar 26/10/10	mar 02/11/10
15	Diseño de la aplicación	1 día	mar 02/11/10	mié 03/11/10
16	Diseño de las pruebas de Test	3 horas	mié 03/11/10	mié 03/11/10
17	Documentación del Diseño	1 sem	mié 03/11/10	mié 10/11/10
18	Aprobación del diseño	1 hora	mié 10/11/10	mié 10/11/10
19	**Desarrollo de la aplicación**	**121 días?**	**mié 10/11/10**	**jue 28/04/11**
20	Preparación del entorno de desarrollo	2 horas?	mié 10/11/10	mié 10/11/10
21	Desarrollo Sprites y Mapas	166 horas	jue 11/11/10	jue 09/12/10
22	Funcionalidades de la aplicación	800 horas	jue 09/12/10	jue 28/04/11
23	**Tests y pruebas**	**6,5 días?**	**jue 28/04/11**	**lun 09/05/11**
24	Tests de funcionamiento	19 horas	jue 28/04/11	mar 03/05/11
25	Documentación de desarrollo	4 días?	mar 03/05/11	lun 09/05/11
26	Aprovación del desarrollo	1 hora	lun 09/05/11	lun 09/05/11
27	**Implantación**	**1 día?**	**lun 09/05/11**	**mar 10/05/11**
28	Instalación	1 hora	lun 09/05/11	lun 09/05/11
29	Pruebas reales	7 horas?	lun 09/05/11	mar 10/05/11
30	Generación de documentos (memoria)	30 días	mar 10/05/11	mar 21/06/11
31	Cierre Proyecto (Entrega Memoria)	1 hora	mar 28/06/11	mar 28/06/11
32	Presentación	6 días?	mié 06/07/11	mié 13/07/11

Figura 3.2: Planificació temporal del projecte.

3.5.3 Diagrama de Gantt del Projecte.

La figura 3.3 mostra el diagrama de Gant previst del projecte (de dreta a esquerra: part1, part2).

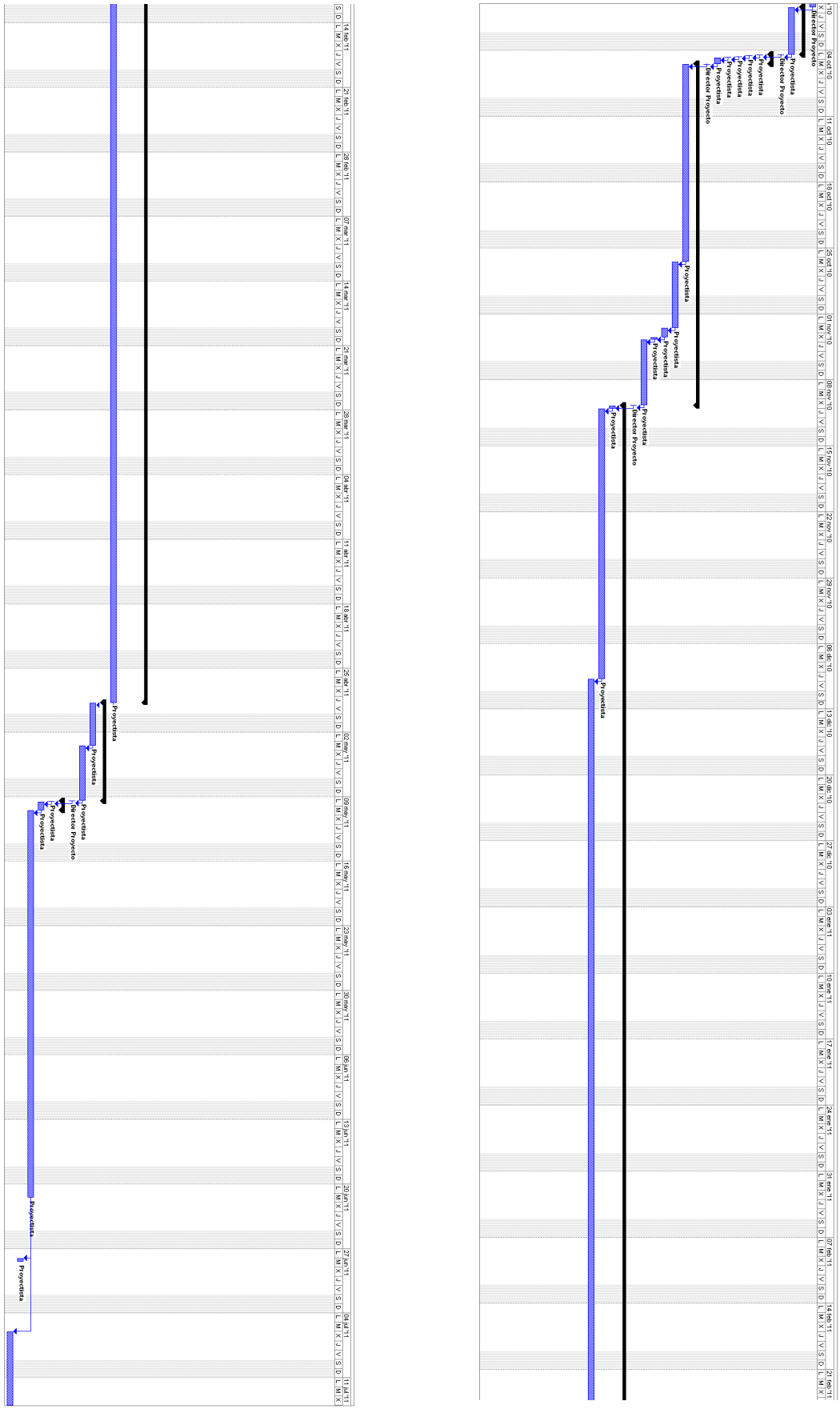


Figura 3.3: Diagrama de Gantt Parts 1 i 2.

3.6 Avaluació de riscos.

3.6.1 Llistat dels riscos.

1.- El llenguatge no incorpora alguna funció o llibreria pròpia que realitzi alguna tasca bàsica per al correcte funcionament del joc.

2.- No s'acaba dins del termini preestablert.

3.6.2 Pla de contingència dels riscos.

1.- Incorporar llibreries externes (no oficials) compatibles i que incloguin aquestes funcions o crear les funcions ajustant les necessitats.

2.- Eliminar alguns requisits funcionals opcionals per fer el projecte més viable (eliminar estats innecessaris que únicament aportin varietat de moviments a més dels bàsics i millores visuals prescindibles a la interfície).

3.7 Conclusions de l'estudi de viabilitat.

3.7.1 Beneficis:

1.- Estudi en el desenvolupament d'un joc.

2.- Estudi de les maneres d'aplicar les diferents tècniques de desenvolupament **d'intel·ligència artificial** dins del camp dels videojocs, que es troba en constant evolució.

3.- Estudi i implementació de la concurrència entre processos per sincronitzar-los.

4.- Proporciona entreteniment a l'usuari final amb pocs requisits (com per exemple els especificats a l'apartat de la Descripció física del sistema).

3.7.2 Inconvenients:

1.- Llenguatge relativament recent en el qual podrien faltar algunes utilitats que ha d'implementar el programador.

2.- No és directament portable a altres sistemes que no estiguin suportats oficialment sense tornar a compilar o sense afegir algunes modificacions (al codi o a la màquina on s'utilitzi).

3.7.3 Conclusions finals de l'estudi:

La **intel·ligència artificial** es pot aplicar a molts camps dins del desenvolupament de software i els videojocs suposen un gran ventall de possibilitats ja que són algunes de les aplicacions més comuns que la inclouen. Per altra banda el llenguatge utilitzat permet prioritzar les funcionalitats del joc i els seus aspectes a alt nivell amb resultats ben satisfactoris, tot i que al ser poc conegut i suposar un tipus de programació diferent (orientada al procés) segurament calgui un temps d'aprenentatge previ.

De tota manera al tractar-se d'un projecte sense cap cost i estèticament senzill es pot dir que el projecte és viable.

4. Anàlisi del Disseny

El projecte té requisits amb diferents nivells de prioritat ja que tot i que a simple vista tots poden semblar igual d'importants a l'hora de realitzar un joc alguns poden establir la diferència entre un correcte funcionament encara que sigui mínim i una llista d'errors de funcionament.

4.1 Requisits funcionals.

4.1.1 Requisits crítics.

1.- Control dels moviments del personatge introduïts pel jugador.

S'ha d'avaluar si es pressionen determinades tecles (moviment, salt, bloqueig o atac)[16] i fer que el personatge del jugador realitzi les accions corresponents a aquestes tecles.

2.- Control de l'*scroll*.

Per poder garantir la jugabilitat i el correcte funcionament l'*scroll* ha de mantenir sempre en pantalla al personatge del jugador respectant els límits horitzontals i verticals dels escenaris. També ha d'evitar que el desplaçament total del jugador sigui major que les dimensions de l'escenari ja que s'unirien la imatge del final de l'escenari (a la seva direcció vertical o horitzontal) amb el principi del mateix.

3.- Gestionar les vides.

El jugador tindrà un màxim de 3 oportunitats per superar un nivell sense perdre punts, tot i que per afegir alguna dificultat no es podran recuperar. A mesura que el jugador va rebent cops la seva salut va disminuint i un cop arriba a 0 el personatge mor, perd una vida y torna a estar amb la salut recuperada.

4.- Controlar que si no li queden vides s'acaba el joc.

Ja que no hi ha un límit establert de pantalles serà la habilitat del jugador la que determini quan s'acabarà el joc. Un cop perdi totes les vides es perdrà una opció de continuació. Si no queden continuacions es mostra la pantalla de fi del joc.

5.- Gestió de la salut dels enemics.

De la mateixa manera que el personatge del jugador va perdent salut els enemics també la perden a mesura que reben cops i un cop la seva salut arriba a 0 moren i desapareixen.

6.- Sincronització entre els diferents processos per evitar efectes indesitjats.

Tal com s'ha vist al capítol de l'estudi de viabilitat, el llenguatge que s'utilitzarà està orientat als processos de manera que és molt fàcil pensar en que la majoria de processos s'hauran de sincronitzar per evitar que més processos agents del compte realitzin una determinada acció que podria comportar un error del joc o un fet poc creïble (per exemple, dos enemics que ataquin al jugador des de la mateixa posició). Aquest és un dels fets pels quals la **concurrència**

[16] Per defecte les fletxes corresponen al moviment, 'A' correspon a l'atac, 'S' al bloqueig i 'D' al salt.

és un aspecte bastant present la majoria del temps. Però no només això si no que al haver-hi múltiples **agents** a la vegada amb l'objectiu de vèncer al jugador es tracta d'un sistema amb **planificació multiagent**[17][RB2] cooperatiu i la **concurrència** s'utilitza com a mètode de coordinació.

7.- Gestionar la presa de decisions dels enemics.

Per facilitar la comprensió i garantir a més a més un bon rendiment tenint en compte totes les possibles decisions dels agents i les reaccions automàtiques involuntàries (tant dels agents com del jugador) s'han utilitzat màquines d'estats finits.

8.- Controlar les dureses.

Ja que els escenaris son imatges que han de tenir un suposat efecte de realisme pel que fa a la consistència dels elements sòlids del fons s'ha de procurar que d'alguna forma els agents (i el jugador) no puguin travessar aquests elements de l'escenari com si no hi fossin. D'aquesta manera, per exemple no es poden ignorar les parets o el propi terreny.

9.- Gestió de les col·lisions i la física[18][RB9] **del joc.**

De la mateixa manera que els escenaris tenen els seus elements sòlids que no es poden travessar, els **agents** i el personatge del jugador també son sòlids i per tant no poden existir al mateix temps al mateix espai i per norma general estan sotmesos a la llei de gravetat, de manera que per norma general no poden "volar" per l'escenari. De la mateixa manera si reben un cop han de reaccionar com a conseqüència.

10.- Gestió dels escenaris.

El joc està dividit en nivells, cadascun de les quals inclou un escenari. Els escenaris estan dividits en diverses parts que s'han de superar per poder superar el nivell. Un cop es supera el final d'un escenari s'ha d'inicialitzar el següent.

4.1.2 Requisits de prioritat alta.

1.- Gestió de la divisió dels escenaris en diverses parts.

Tots els escenaris tenen un límits preestablerts que el jugador ha d'anar superant per poder acabar amb èxit el nivell.

2.- Gestionar els límits parcials, inicial i final dels escenaris.

Els escenaris estan dividits en parts i l'escenari és qui s'encarrega d'afegir els enemics en funció de la dificultat.

[17] Es tracta d'un entorn en el qual hi ha diversos agents que poden tenir un mateix objectiu comú: compartir (cooperatiu) o poden tenir un objectiu basat en maximitzar el rendiment propi minimitzant el d'altres agents (competitiu).

[18] Lleis de la física aplicades al joc.

3.- Evitar que el jugador travessi els límits dels nivells sense complir el requisit previ.

El jugador no pot passar per un límit sense haver eliminat els enemics que encara siguin vius i dins del límit. De la mateixa manera l'*scroll* tampoc podrà avançar més enllà dels límits corresponents, de manera que indica al jugador fins a on pot avançar (a més del corresponent indicador per continuar avançant). A més el límit esquerre generalment sempre coincideix amb l'extrem esquerre de la pantalla (o finestra) del joc.

4.- Controlar la salut del personatge principal i mostrar-la en tot moment.

A mesura que va rebent cops la salut del personatge va disminuint i es mostra com una barra indicativa a la part superior de la interfície del joc, junt amb la quantitat de vides restants i la puntuació actual.

5.- Mostrar canvis físics als enemics que els hi quedi poca vida.

Quan els enemics tenen poca salut canvien el seu aspecte per un altre amb ferides visibles.

6.- Gestió del temps restant per superar un dels límits.

A més dels enemics el factor del temps també és important a l'hora de superar un límit de l'escenari ja que el joc disposa d'un temporitzador per a cada límit. Si el jugador supera el límit abans de que acabi el temps el temporitzador torna a inicializar-se i en cas contrari mor. A més cada cop que el personatge mor també s'inicialitza el temporitzador.

7.- Gestionar la quantitat de reaccions possibles dels agents segons la dificultat.

La dificultat no només afecta a la quantitat d'enemics que hi ha per nivell si no que també influeix en la capacitat que tenen els enemics per prendre decisions. A mesura que augmenta la dificultat del joc també augmenten la quantitat d'accions que poden fer els agents ja sigui a nivell d'atacs o a nivell estratègic (per exemple si s'ha arribat a una determinada dificultat i un enemic veu que té poques possibilitats pot optar per fugir).

8.- Gestionar la puntuació durant i al final del nivell.

A mesura que el jugador va eliminant enemics va augmentant la seva puntuació i un cop superat el nivell rep una sèrie de bonificacions en funció del nivell superat. Però si mor, es queda sense vides i li queden continuacions reapareixerà però la puntuació es veurà greument disminuïda (dividida per la meitat).

9.- Gestionar els canvis de dificultat.

Un cop finalitzat un nivell i calculada la puntuació total, abans d'inicialitzar el següent nivell es tindrà en compte aquesta dificultat per augmentar o disminuir la dificultat del joc.

4.1.3 Requisits opcionals.

1.- Enemics ocults segons la dificultat.

Als nivells poden haver-hi objectes que poden amagar algun enemic i si s'arriba a una certa dificultat aquests enemics poden aparèixer.

4.2 Requisits no funcionals.

4.2.1 Requisits crítics.

1.- Els recursos (escenaris i *sprites*) han de ser de creació pròpia.

Per garantir una originalitat pròpia i evitar el risc de violar drets d'autor d'altres *sprites* i mapes tots aquests recursos estaran fets a mà amb *The GIMP.*

4.2.2 Requisits de prioritat alta.

1.- La interfície ha d'informar de tot el que calgui sense incomodar al jugador.

Per garantir una bona jugabilitat la interfície del joc ha de ser clara, informant de tots els elements importants (salut, puntuació i vides) sense incomodar al jugador i sense que distregui la atenció de l'acció principal.

2.- Ha de consumir pocs recursos.

El joc ha de minimitzar l'ús de recursos innecessaris tant a nivell de recursos físics (imatges) com a nivell de programació (per exemple massa càrrega d'imatges i processos a la memòria o l'ús indiscriminat de funcions i variables poc o gens utilitzades).

4.2.3 Requisits opcionals.

1.- Fer un port per alguna consola compatible (per exemple Wii, PSP o GP2X).

5. Disseny del joc

5.1 Interfície del joc.

La interfície d'un joc desenvolupa un paper més important del que sembla a simple vista ja que no només ha d'informar al jugador de tot el que sigui necessari ja sigui en un moment donat o durant tota la partida sinó que també ha de ser clara i ha d'evitar distreure l'atenció del jugador de l'acció principal (els enemics que intenten abatre al jugador) tot el que sigui possible sense arribar a molestar (interfície massa vistosa o massa carregada d'informació). Per aquests motius s'ha optat per un model simple que agrupa tota la informació que es mostra contínuament a la part superior de la interfície mentre que la resta de la informació (com ara els diferents indicadors) es mostra només quan no suposa cap inconvenient al jugador, aportant un detall estètic a més de la informació.

El projecte conté tres interfícies: la interfície de l'acció del joc, la interfície de nivell superat i la interfície de final del joc. Totes estan a una resolució de 800x600 en mode finestra principalment per fer-lo compatible amb targetes gràfiques menys potents.

La figura 5.1 mostra la interfície bàsica de l'acció del joc.

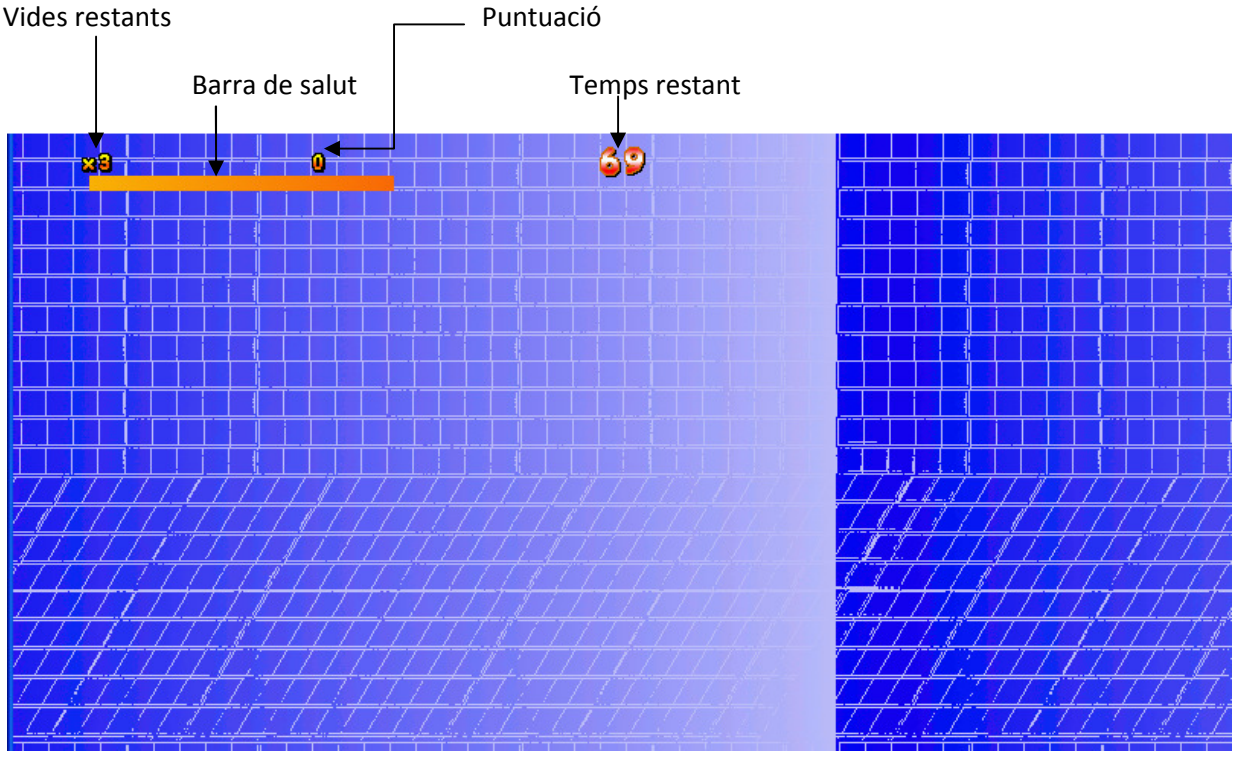


Figura 5.1 Interfície del joc

La interfície de final de nivell mostra la puntuació i un missatge amb la tecla que cal polsar per continuar.

La interfície de fi del joc només mostra el text "GAME OVER" sobre un fons negre.

5.2 La intel·ligència artificial.

Un dels aspectes més elaborats pel que fa al desenvolupament d'un videojoc (i a aquest projecte) és precisament el de la **I.A.** per la seva complexitat i diferents formes d'afrontar un mateix problema. Cada tipus de joc s'adapta millor a un determinat mètode de desenvolupament de la **I.A.** però a més cal tenir en compte la quantitat d'**agents** que coexisteixen a la vegada. Tenint en compte aquests dos factors i el tipus de joc a desenvolupar (joc d'acció a temps real amb un **entorn multiagent**) s'ha optat per separar la gestió de les **decisions i reaccions** dels agents de la **presa de decisions** i tractar-los de forma independent. La **gestió de les decisions** i **les reaccions** es fa mitjançant **FSM**s tant per als enemics i objectes com per al propi jugador sempre que calgui. La diferencia principal entre els agents enemics i objectes i el jugador és que pel cas del jugador s'han de combinar amb les accions corresponents als botons premuts (segons s'indica al capítol anterior).

La presa de decisions es duu a terme mitjançant mètodes de concurrència aprofitant l' **entorn multiagent** i que pel propi llenguatge utilitzat els **agents** es poden tractar com a **processos independents**.

5.2.1 Màquines d'Estats Finits (FSM).

Tots els **agents** (protagonista, enemics i objectes) funcionen principalment gracies a una **FSM** que relaciona tots els seus possibles estats (voluntaris o involuntaris). En el cas del protagonista aquesta màquina d'estats es combina amb els controls del jugador. En qualsevol cas s'ha triat el model de la **Màquina de Moore**[Anex2] [RB2] per simplicitat tant a l'hora de programar com per entendre el seu funcionament de manera que els estats de la màquina coincideixen amb els estats de l'**agent** i les transicions indiquen la condició que s'ha de dur a terme per canviar a un estat determinat. La figura 5.2 fa referència a la **FSM** que s'ha utilitzat per als enemics distingint amb colors la dificultat necessària per a que es pugui accedir a les corresponents transicions. Els estats més destacats son:

- Huyendo: Fa que un enemic pugui fugir per tenir més possibilitats més endavant.
- Aproximandose para atacar: Permet triar la manera en la s'acostarà al jugador per atacar i la posició des de la que atacarà.
- Vigilando la zona: Tracta d'evitar que el jugador pugui apartar-se dels enemics que l'envolten.
- Esperando: Espera a que el jugador se li acosti o fugi abans d'entrar en acció.
- Decidiendo ataque: Un cop davant del jugador permet triar l'atac entre un normal o un altre més potent.

Al capítol de la implementació s'expliquen amb detall tots els estats.

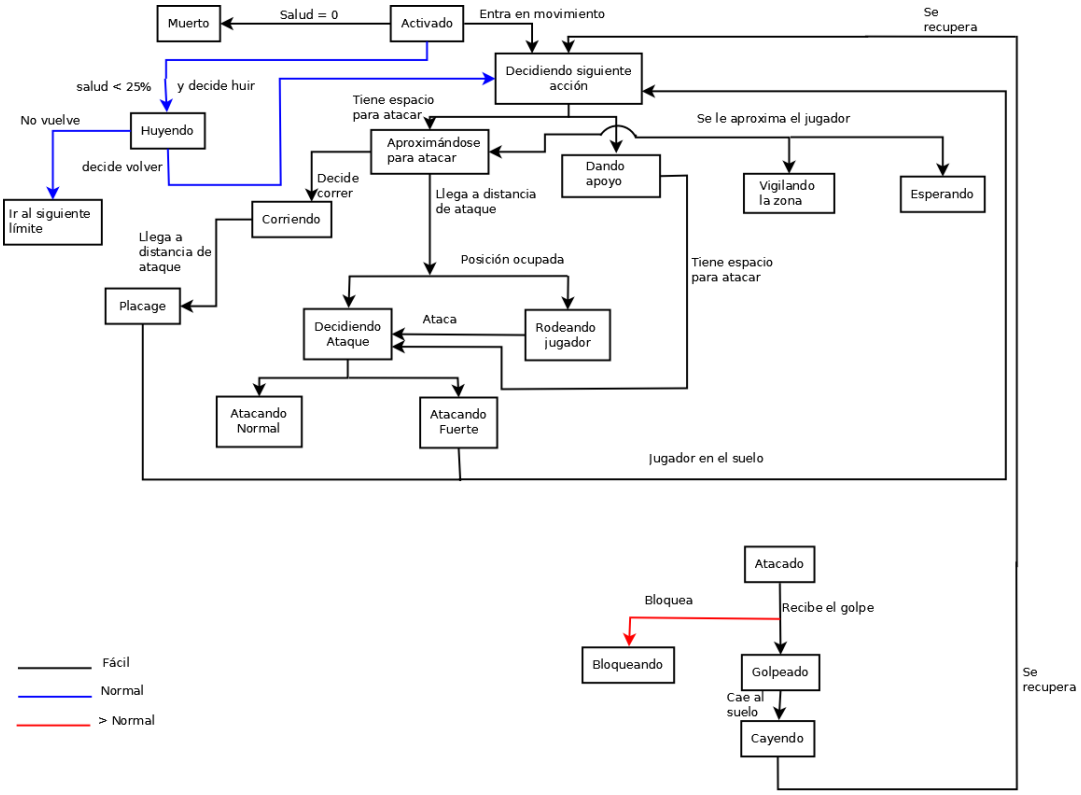


Figura 5.2: Màquina d'Estats Finits d'un enemic.

Cal tenir en compte que les possibilitats d'accedir a un estat o un altre (entre els disponibles) poden variar segons la dificultat del joc.

5.2.2 Entorn multiagent.

Tal i com s'ha dit al punt 5.2 el joc generalment té més d'un **agent** funcionant a la vegada i això significa que cal fer que "es posin d'acord" a l'hora de realitzar les seves accions. Això permet sincronitzar-los entre ells per evitar que tots es comportin de la mateixa manera provocant un comportament poc intel·ligent o el que és pitjor, produir un mal funcionament del joc. En el cas d'un **entorn multiagent** cooperatiu com és aquest cas (els enemics cooperen entre ells) un dels recursos més utilitzats per aconseguir aquesta "cooperació" es mitjançant la **concurrència**.

En aquest cas s'han utilitzat **semàfors** comptadors, que tot i que el llenguatge no els permet definir directament (sense alguna llibreria externa) es poden crear fàcilment tenint en compte que no han d'aturar un procés completament (en espera) sinó que únicament han de triar un estat de l'agent o un altre depenent de les condicions prèvies (transició de la **FSM**, dificultat o si l'estat està disponible). És a dir, que es poden fer servir mitjançant comptadors que excloguin els estats necessaris.

A continuació s'exposa el problema junt amb la solució optada per a facilitar la comprensió del que s'ha explicat abans.

5.2.2.1 Problema sincronització dels enemics.

Cada enemic té els següents estats a tractar (únicament s'inclouen els estats que fan referència a aquest problema):

- APROX_JUGADOR: Aproximar-se per atacar.
- ESPERANT: Espera a que el jugador se li aproximi.
- AJUT: S'acosta al jugador sense atacar donant reforç als seus companys. Si el jugador se li acosta, ataca.
- VIGILANT: Camina per l'escenari per tallar possibles vies d'escapament del jugador.

Hi ha un número específic d'agents que poden accedir a la majoria d'aquests estats:

- n_esperant=N.
- n_vigilant=M.

On N i M son números enters (entre 1 i 3) que es corresponen amb el número màxim d'enemics que poden fer la acció corresponent.

A més únicament hi pot haver un enemic atacant al jugador per cada posició (pel davant o per darrera) i per accedir a AJUT directament (encara que hi hagi possibilitats de fer qualsevol altra acció) cal que hi hagi una dificultat mínima. Per triar els estats de forma més imprevisible es fa servir a més a més un factor aleatori.

El funcionament és el següent:

- Si no hi ha cap enemic atacant llavors l'enemic ataca ocupant una de les dues posicions disponibles i si vol atacar un altre enemic llavors es situa a la posició disponible.
- Si estan totes les posicions d'atac ocupades i hi ha un nivell de dificultat igual o superior al 'Normal' l'enemic pot accedir a la resta d'estats sempre i quan s'hi pugui accedir. Si no es pot accedir passa a AJUT. Si la dificultat és menor del necessari tria un dels estats disponibles (VIGILANT o ESPERANT) i si s'ha arribat al seu màxim (M o N respectivament) mirar si l'altre estat està disponible i com a últim cas tria l'estat AJUT.
- Independentment de la dificultat cal tenir en compte que tot i que pot haver un enemic que no estigui atacant encara pot haver-hi algun a l'estat APROX_JUGADOR de manera que cal tenir en compte aquest fet per evitar errors.

Suposem un exemple amb dificultat 'Normal' en el qual hi ha E enemics (E1, E2, E3, E4 i E5) i tant M com N valen 1.

- L'enemic E1 ataca ja que té posicions disponibles (no n'hi ha cap ocupada) i decideix atacar per darrera (passa a APROX_JUGADOR).
- E2 també pot atacar i només pot fer-ho per davant (passa a APROX_JUGADOR).
- E3 decideix quedar-se esperant aprofitant que encara no hi ha cap enemic fent-ho (passa a ESPERANT, n_esperant passa a ser 1, que és igual que N).
- E4 intenta atacar també però com totes les posicions estan ocupades decideix vigilar la zona per si el jugador es vol escapar (passa a VIGILANT, n_vigilant passa a ser 1, que és igual que M).

- E5 vol vigilar també però com que n_vigilant ja és al seu valor màxim (M) no ho pot fer. Finalment decideix ajudar als seus companys (passa a AJUT).

5.2.2.2 Solució del plantejament.

Un cop el problema està plantejat es pot veure com té algunes similituds amb el problema de concurrència del Barber[RB11] tot i que s'hi afegeixen altres condicions i un factor d'aleatorietat.

El primer pas es comprovar si hi ha algun enemic acostant-se al jugador o atacant-lo per la dreta o l'esquerra. Si l'enemic pot acostar-se al jugador llavors tria una de les opcions a l'atzar sempre i quan l'acció estigui disponible (el número d'enemics realitzant aquesta mateixa acció sigui menor al seu màxim corresponent). Si no pot realitzar-la passa a ajudar als seus companys. Si directament no pot acostar-se al jugador llavors tria una de les altres accions sempre que pugui però sense tenir en compte la opció d'acostar-se per atacar.

En el cas d'una dificultat 'Fàcil' el funcionament es similar al que s'ha plantejat anteriorment per a una dificultat que sigui almenys 'Normal' però es redueix el pes que té l'aleatorietat reduint els casos a triar i un cop comprovat un dels casos aleatoris la resta de la tria és seqüencial sempre i quan els enemics puguin realitzar la acció avaluada.

Al capítol de la implementació es fa una explicació més detallada d'aquest funcionament

5.3 Escenaris.

Els escenaris tot i tenir una perspectiva lateral disposen de profunditat de moviment, de manera que el jugador i els enemics poden desplaçar-se en qualsevol direcció sempre i quan no hi hagi cap impediment pel terreny com les parets. A més el desplaçament del jugador estarà també limitat als límits dels escenaris que ha d'anar superant i l'*scroll* el manté sempre en pantalla.

6. Implementació

A continuació es fa una descripció de la implementació del projecte a nivell funcional tant pel que fa al funcionament de les **FSMs** (dels enemics, jugador i objecte), la coordinació entre els enemics, la gestió dels escenaris, funcions i els diferents estats dels **agents**.

6.1 FSM.

Per a la implementació de les Màquies d'estats finits s'ha aprofitat el propi funcionament d'un *switch*[19][RB27] fent servir els diferents casos (*case*) com als estats corresponents a cada agent. De fet Brian Schwab descriu al seu llibre *AI Game Engine Programming*[RB1] al switch com a exemple més bàsic d'una màquina d'estats finits. La forma en la que es gestionen els canvis d'estats és diferent segons el tipus d'**agent** (enemic, protagonista o objecte) ja que el seu funcionament és diferent ja sigui a nivell de complexitat o a nivell de forma d'avaluar els estats (en el cas del protagonista). A continuació s'explica la implementació per a cada tipus d'agent.

6.1.1 FSM dels enemics.

Per gestionar els canvis d'estats s'ha fet servir la funció **cambia_estado** que rebent una ID y el nou estat s'encarrega de fer la part corresponent a la comunicació de l'**entorn multiagent** incrementant i decrementant els corresponents comptadors, actualitza posicions si cal li canvia l'estat.

6.1.1.1 Funció cambia_estado (int procesoID, string nuevoEstado).

En primer lloc aquesta funció s'encarrega de la comunicació amb la resta dels enemics:

- Primerament comprova si la variable **nuevoEstado** correspon als estats VIGILANDO, ESPERANDO, o APROX_JUGADOR i si és el cas incrementa el seu comptador corresponent. Seguidament comprova l'estat actual del procés (enemic) i si es correspon a VIGILANDO, ESPERANDO, o APROX_JUGADOR decrementa el seu comptador corresponent.
- Quan l'enemic deixa d'atacar i es retira (o cau a terra), s'allibera la seva posició per a que sigui accessible per altres enemics.

Seguidament inicialitza un temporitzador per evitar que un enemic s'estigui quiet o esperant indefinidament.

Per acabar guarda l'estat actual d'aquest procés enemic i finalment actualitza el seu nou estat.

6.1.2 FSM del protagonista.

El funcionament d'aquesta Màquina d'estats finits és notablement diferent al de la resta ja que per poder garantir el seu funcionament correcte a l'hora de prémer les tecles adequades no es poden tractar els estats directament sense tenir altres aspectes en compte. Bàsicament el procés del protagonista utilitza el **polling**[20][RB28] comprovant si hi ha una tecla concreta pitjada i

[19] En programació és una estructura de control com poden ser *if* o *if else* en les que s'avaluen unes condicions. Es pot fer servir com a múltiples sentències *if* anidades.

[20] En general fa referència a una operació de consulta constant, per crear una activitat sincrònica sense l'ús d'interrupcions, generalment cap a un dispositiu hardware, tot i que també es pot fer servir per a recursos software.

llavors fa les accions corresponents. Això inevitablement comporta un cas especial de concurrència ja que si no es vol afegir un bucle per a cada acció des de que comença l'acció fins al final de la mateixa s'han de tractar aquestes accions com a seccions crítiques del propi procés. D'aquesta manera s'aconsegueix mitjançant variables de control que utilitzant únicament la iteració del bucle principal es mantingui l'estat de l'acció corresponent que s'estigui duent a terme, assegurant la permanència de l'aquest estat fins que aquest acabi.

La resta d'accions automàtiques que no impliquen les accions del jugador directament (caigudes, mort o rebre un atac) es tracten de forma similar a la explicada anteriorment ja que encara que sigui per causes alienes al control del jugador els estats s'han de poder mantenir igualment, almenys fins al final de la seva animació corresponent. Aquest es un fet que s'ha de tenir en compte també ja que per defecte si no es pressiona cap tecla d'acció el personatge del jugador es manté al seu estat d'espera (QUIETO). Més endavant, en aquest capítol s'explicarà amb detall el funcionament general del protagonista.

6.1.3 FSM de l'objecte.

Tot i que possiblement no es pugui considerar un **agent** intel·ligent pròpiament dit, pel que fa a la implementació s'ha dut a terme de la mateixa manera que la resta d'**agents**. En aquest cas l'únic objecte consisteix en una porta (que forma part de l'escenari) amb dos estats (oberta o tancada) que canvien directament (sense cap funció intermediària).

6.2 Els agents.

A continuació es fa una breu explicació de la implementació dels tres tipus d'**agents** que disposa el projecte (enemic, protagonista i objecte) sense entrar en detall en la implementació de les diferents màquines d'estats però si en els seus diferents estats. Tots ells son processos amb un bucle infinit que únicament acaba quan s'indica explícitament (forçant la sortida del bucle o matant el procés). Aquests processos tenen unes variables predefinides (anomenades variables locals) que per propietats del llenguatge permeten ser accessibles per altres processos i funcions com si fossin simples variables globals però independents per a cada procés. Les més útils d'aquestes variables son les que fan referència a la seva posició a la pantalla (x,y) i a la seva profunditat dins de l'escenari (z) ja que el posicionament dels elements (fons, personatges i objectes) a nivell estètic utilitza diferents capes. A més aquests tipus de variables es poden fer servir mitjançant la herència entre processos.

6.2.1 Enemics.

El projecte només disposa d'un tipus d'enemic que inicialment defineix els *sprites* corresponents a les seves animacions inicials (sense estar ferit) i defineix el seu objectiu (el jugador). Després comença el seu veritable funcionament comprovant que hi ha almenys un camí fins al protagonista i gestionant els seus estats per mitjà de la seva **FSM**.

6.2.1.1 Estats dels enemics.

INACTIVO:

Estat per defecte dels enemics un cop es creen.

- Comprova la posició del jugador i si cal l'enemic gira de manera que se'l quedi mirant.
- Espera a que l'escenari estigui inicialitzat si encara no ho està i passa a activar-se (canvia l'estat a ACTIVO).

QUIETO:

- Comprova si cal canviar el seu aspecte pel de ferit. La figura 6.1 mostra la diferència entre l'aspecte normal i el ferit.
- Comprova la seva posició amb la del jugador i si cal es gira.
- Si la distancia entre ell i el jugador és dins d'un rang determinat (entre 50 i 100 píxels), la diferència d'alçades no supera els 10 píxels (estan gairebé a la mateixa posició a l'eix y) i no hi ha cap altre enemic atacant, llavors aquest passa a atacar.
- Si la distancia (tant en longitud com en alçada) no és dins dels marges i no col·lisiona (es topa) amb un altre enemic passa a l'estat actiu (ACTIVO) si col·lisiona passa a ajudar als companys (estat APOYANDO).
- Inclou un temporitzador per evitar que estigui constantment dins d'aquest mateix estat. Si es supera el temps el seu estat passa a ser ACTIVO.

Figura 6.1: Enemic sa i enemic ferit.

ACTIVO:

- Assigna els *sprites* corresponents a la animació si encara no ho ha fet.
- Tria la seva estratègia amb la funció **set_estadoProximaPosicion(int factor, int procesoID)**.

ESPERANDO:

- Comprova si cal canviar el seu aspecte pel de ferit. La figura 6.1 mostra la diferència entre l'aspecte normal i el ferit.
- Si la distancia entre ell i el jugador és menor de 100 píxels tria entre adoptar una nova estratègia amb la funció **set_estadoProximaPosicion(int factor, int procesoID)** o apartar-se i esperar una oportunitat millor amb la funció **get_destino_aprox(int coord, string tmpestado,int IDobjetivo)**.
- Comprova que sempre estigui mirant al jugador.
- Inclou un temporitzador per evitar que estigui constantment dins d'aquest mateix estat. Si es supera el temps el seu estat passa a ser ACTIVO.

VIGILANDO:

- Calcula un destí pròxim al jugador per acostar-se mantenint una certa distància comprovant que aquest destí no sigui part d'alguna duresa de l'escenari.

- Canvia l'estat a CAMINANDO amb la funció **cambia_estado (int procesoID, string nuevoEstado)**.

APOYANDO:

- Comprova que la distancia entre ell i el protagonista sigui major o igual a 100 píxels i si no és així canvia el seu estat a QUIETO amb la funció **cambia_estado (int procesoID, string nuevoEstado).**
- Si la distancia és igual o major a 20 píxels (i menor que 100) i el jugador no té cap altre enemic al costat (davant o darrera) tria una altra acció amb la funció **set_estadoProximaPosicion(int factor, int procesoID).**
- Si col·lisiona amb un altre enemic o amb el jugador i està a una alçada similar tria una posició propera per apartar-se. Si no, tria un destí pròxim al jugador comprovant que aquest destí no formi part duna duresa de l'escenari i canvia l'estat a CAMINANDO.

APROX_JUGADOR:

- Únicament tria si l'enemic s'acosta al jugador caminant o corrents amb la funció **set_estadoAproximacion(int objetivoID, int procesoID)**.

CAMINANDO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat amb el procés **animar (int primero , int ultimo)**.
- Comprova quin ha estat el seu estat anterior i si ja ha arribat al seu objectiu. Si l'estat era VIGILANDO o APOYANDO torna a l'estat anterior. Si l'estat era HUYENDO o ESPERANDO canvia a l'estat ESPERANDO.
- Mentre no arribi a la posició objectiu es va acostant.
- Si el jugador se li apropa abans d'arribar a la posició objectiu passa a l'estat QUIETO per poder reaccionar amb una altra estratègia.
- Si col·lisiona amb un altre enemic o el protagonista canvia a l'estat RODEANDO o bé QUIETO si el col·lisionat (en el cas d'un company enemic) ja està rodejant a algú.
- Evita que l'enemic avanci a través de les dureses rodejant-les si cal.

CORRIENDO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat amb el procés **animar (int primero , int ultimo)**.
- Si la distancia amb el jugador és menor de 200 píxels pot triar a fer un placatge al jugador canviant a l'estat PLACANDO o continuar corrents cap al jugador.
- Si col·lisiona amb un altre enemic o el protagonista canvia a l'estat RODEANDO o bé QUIETO si el col·lisionat (en el cas d'un company enemic) ja està rodejant a algú.
- Evita que l'enemic avanci a través de les dureses rodejant-les si cal.

La figura 6.2 mostra un grup d'enemics corrents cap al jugador.

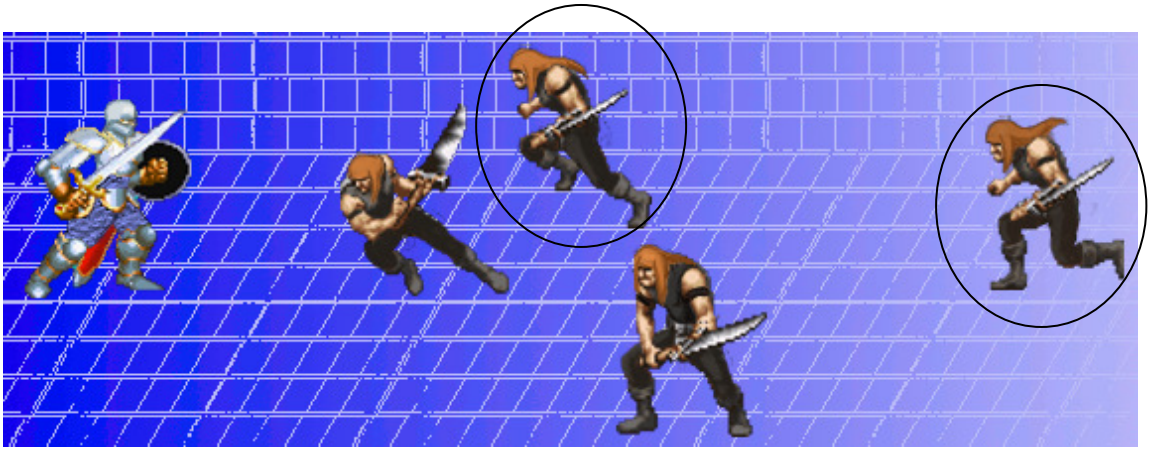


Figura 6.2: Grup d'enemics corrents cap al jugador.

RODEANDO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat amb el procés **animar (int primero , int ultimo)** depenent de si l'estat anterior era CAMINANDO o CORRIENDO.
- Rodeja a l'enemic o al jugador evitant el desplaçament per les dureses del terreny.
- Si torna a col·lisionar amb un altre enemic o amb el protagonista (si abans no ho havia fet) també el rodeja.
- Un cop acaba de rodejar torna al seu estat anterior.
- Si surt del límit de la pantalla torna a entrar per la mateixa banda (dreta o esquerra) però a una alçada diferent tenint en compte les dureses de l'escenari.

HUYENDO:

- Tria entre fugir de l'escena i esperar al jugador a un altre secció del nivell (si no està ja a la última secció) o apartar-se esperant a que els altres enemics deixin més tocat al jugador.
- Canvia a l'estat CAMINANDO.

ATACADO:

- Gestiona les reaccions de l'enemic en cas de ser atacat. Segons la dificultat del joc pot tenir més possibilitats de bloquejar l'atac passant a l'estat BLOQUEANDO. Si no el bloqueja passa a l'estat GOLPEADO.

GOLPEADO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat.
- Disminueix la seva salut i si arriba a 0 mor (canvia a l'estat MUERTO).
- Si rep un atac fort cau a terra (canvia a l'estat CAYENDO).

La figura 6.3 mostra a un enemic rebent un atac sense poder bloquejar-lo.

Figura 6.3: Enemic rebent un atac.

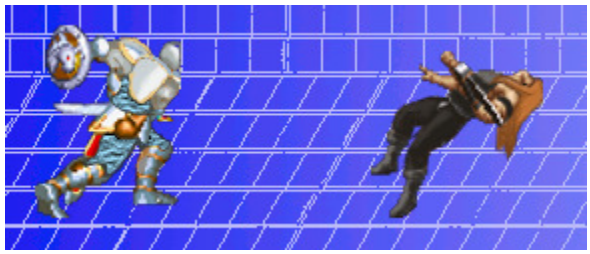
BLOQUEANDO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat.
- Tria entre estar-se quiet (canvia l'estat a QUIETO) o contraatacar (canvia l'estat a APROX_JUGADOR).

CAYENDO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat. En aquest cas l'*sprite* només es modifica quan el comptador assignat per a la caiguda ho indica.
- L'agent es desplaça cap enrere mentre va caient i finalment es desplaça cap avall fins que arriba al terra (segons la seva alçada anterior).
- Si la salut és menor d'un 25% i la dificultat és com a mínim 'Normal', pot fugir.

La figura 6.4 mostra un enemic caient per culpa d'un atac fort.



6.4: Enemic caient.

MUERTO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- El funcionament és similar a l'estat CAYENDO però en comptes de recuperar-se es queda a terra mort fins que desapareix.
- S'incrementa la puntuació del jugador i es decrementa el número d'enemics que queden a la corresponent secció del nivell.
- S'avisa a un altre enemic amb la funció **avisaProc(int IDexcluida)**.
- Finalment el procés es manté dins d'aquest estat fins que rep un senyal (KILL) per matar-lo un cop es supera una secció del nivell.

PLACANDO

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat.
- Comprova el sentit que ha de tenir el placatge (dreta o esquerra).
- Si impacta amb el jugador i aquest no el bloqueja rep un cop fort (força de l'agent x 2), perd vida i cau a terra.

La figura 6.5 mostra a un enemic fent un placatge al jugador mentre un altre aprofita per situar-se en millor posició.

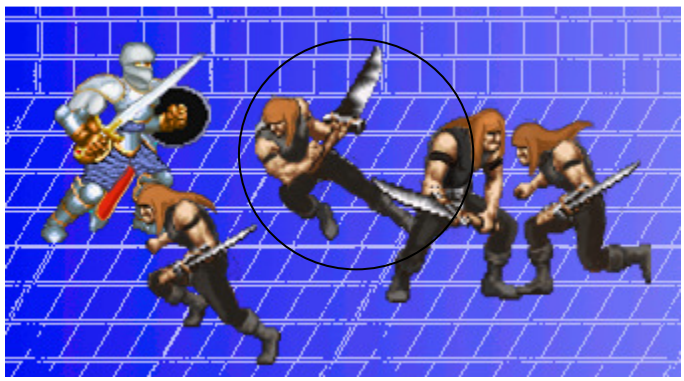


Figura 6.5: Enemic placant.

ATACANDO:

- Comprova la seva salut per si ha de canviar els *sprites* per els que està ferit.
- Gestiona la animació corresponent a aquest estat amb un comptador que indica l'inici i el final de l'atac.
- Comprova la seva posició respecte al jugador per encarar-se.
- Assigna la posició de l'atac respecte del jugador (dreta a esquerra) com a ocupada per evitar que altres enemics intentin atacar des d'aquest punt.
- Si el jugador no bloqueja l'atac, perd vida (força de l'enemic).
- Un cop finalitza l'atac l'estat passa a ser QUIETO per deixar a un altre company (si n'hi ha) atacar y tornar a atacar ell.

La figura 6.6 mostra un enemic atacant i al jugador rebent el cop.

Figura 6.6: Enemic atacant

6.2.1.2 Funcionament de les decisions de l'enemic.

Al capítol del disseny s'ha parlat de manera introductòria del funcionament que tenen els enemics a l'hora de triar la forma d'atacar al jugador plantejant el problema i donant una possible solució però sense entrar en detalls amb l'algorisme. Dins d'aquest apartat es parla precisament d'aquest algorisme.

Si es torna a plantejar el problema de la mateixa manera:

Cada enemic té els següents estats a tractar:

- APROX_JUGADOR.
- ESPERANT.
- AJUT.
- VIGILANT.

Hi ha un número específic d'enemics que poden accedir als estats ESPERANT o VIGILANT:

- n_esperant=N.
- n_vigilant=M.

On N i M son números enters (entre 1 i 3) que es corresponen amb el número màxim d'enemics que poden fer la acció corresponent. També hi ha un màxim d'enemics que poden anar cap al jugador a la vegada.

A més únicament hi pot haver un enemic atacant al jugador per cada posició (pel davant o per darrera) i per accedir a AJUT directament (encara que hi hagi possibilitats de fer qualsevol altra acció) cal que hi hagi una dificultat mínima. Per triar els estats es fa servir a més a més un factor aleatori.

El funcionament és el següent:

- Si no hi ha cap enemic atacant llavors l'enemic ataca ocupant una de les dues posicions disponibles i si vol atacar un altre enemic llavors es situa a la posició disponible.
- Si estan totes les posicions d'atac ocupades i hi ha un nivell de dificultat igual o superior al 'Normal' l'enemic pot accedir a la resta d'estats sempre i quan s'hi pugui accedir. Si no es pot accedir passa a AJUT. Si la dificultat és menor del necessari tria un dels estats disponibles (VIGILANT o ESPERANT) i si s'ha arribat al seu màxim (M o N respectivament) mirar si l'altre estat està disponible i com a últim cas tria l'estat AJUT.
- Independentment de la dificultat cal tenir en compte que tot i que pot haver un enemic que no estigui atacant encara pot haver-hi algun a l'estat APROX_JUGADOR de manera que cal tenir en compte aquest fet per evitar errors.

Es pot fer servir l'algorisme de la figura 6.7 per tractar el problema tant per a una dificultat 'Normal' o bé l'algorisme de la figura 6.8 per fer-ho amb una dificultat 'Fàcil',tenint en compte que la dificultat està definida entre rangs de números enters (0-2: Fàcil, 3-5: Normal, 6-9: Difícil). Cal mencionar que a la implementació final tots dos algorismes formen part de la mateixa funció, ja que contempla els dos casos.

```
1.  Si ((posició_esquerra=0 or posición_dreta=0) and aproximant-se<1) {
2.          Estat(APROX_JUGADOR)
3.  } Si no{
4.          Si(dificultat >= Normal){
5.                  Si (aproximant<limit_aproximant){
6.                          Si (decisió=0){
7.                                  Si( esperant<N ){
8.                                          Estat(ESPERANT)
9.                                  }Si no {
10.                                         Estat(AJUT)
11.                                 }
12.                         Si(decisió=1){
13.                                 Estat(AJUT)
14.                         }
15.                         Si (decisió=2){
16.                                 Si( vigilant<M ){
17.                                         Estat(VIGILANT)
18.                                 }Si no {
19.                                         Estat(AJUT)
20.                                 }
21.                         } Si no{
22.                                 Estat(APROX_JUGADOR)
23.                         }
24.                 Si no{
25.                         Si (decisió=0){
26.                                  Si( esperant<N ){
27.                                         Estat(ESPERANT)
28.                                 }Si no {
29.                                         Estat(AJUT)
30.                                 }
31.                         Si(decisió=1){
32.                                 Estat(AJUT)
33.                         }
34.                         Si (decisió=2)
35.                                 Si( vigilant<M ){
36.                                         Estat(VIGILANT)
37.                                 }Si no {
38.                                         Estat(AJUT)
39.                                 }
40.                         }
41.         }
```

Figura 6.7: Algorisme de decisió per a una dificultat 'Normal'.

- Les línies 1-3 comproven que hi ha espai per atacar al jugador.
- La línia 4 comprova el valor de la dificultat i la línia 5 comprova que encara poden anar enemics cap al jugador.
- El següent pas és comprovar la decisió aleatòria de l'enemic i segons quina sigui va a la línia corresponent (línies 6,11, 15 o 21).
- Segons la decisió que hagi pres comprova que encara hi hagi posicions disponibles (esperant<N o vigilant<M) en cas de ser necessari i si no n'hi ha passa a l'estat AJUT.
- Si no poden anar enemics cap al jugador (línia 5) salta a la línia 25 i repeteix el procés anterior però sense tenir en compte la possibilitat d'anar a l'estat APROX_JUGADOR.

```
1.  Si ((posició_esquerra=0 or posición_dreta=0) and aproximant-se<1) {
2.          Estat(APROX_JUGADOR)
3.  } Si no{
4.          Si(dificultat >= Normal){
5.                  Si (aproximant<limit_aproximant){
6.                          Si (decisió=0){
7.                                  Si( esperant<N ){
8.                                          Estat(ESPERANT)
9.                                  }Si no {
10.                                         Si(vigilant < M){
11.                                                 Estat(VIGILANT)
12.                                         }Si no{
13.                                                 Estat(AJUT)
14.                                         }
15.                                 }
16.                         Si(decisió=1 o decisió=2){
17.                                 Si(vigilant < M){
18.                                         Estat(VIGILANT)
19.                         }Si no{
20.                                         Si( esperant<N ){
21.                                         Estat(ESPERANT)
22.                                         }Si no{
23.                                                 Estat(AJUT)
24.                                         }
25.                                 }
26.                         Si no{
27.                                 Estat(APROX_JUGADOR)
28.                         }
29.                 Si no{
30.                         Si (decisió=0){
31.                                 Si( esperant<N ){
32.                                         Estat(ESPERANT)
33.                                 }Si no {
34.                                         Si(vigilant < M){
35.                                                 Estat(VIGILANT)
36.                                         }Si no{
37.                                                 Estat(AJUT)
38.                                         }
39.                                 }
40.                         Si no{
41.                                 Si(decisió=1){
42.                                         Si(vigilant < M){
43.                                                 Estat(VIGILANT)
44.                                         }Si no{
45.                                                 Estat(AJUT)
46.                                         }
47.                                 }
48.                         }
49.                 }
50.         }
```

Figura 6.8: Algorisme de decisió per a una dificultat ‘Fàcil’.

- La principal diferència amb el cas anterior és que si l'enemic vol esperar o vigilar (línies 6 i 16) i no pot fer-ho per què ja hi ha el màxim d'enemics fent la mateixa acció comprova si pot fer l'altra acció abans d'anar a l'estat AJUT (línies 7-15, 17-25, 31-39 i 42-46).
- No es té en compte la possibilitat d'entrar a l'estat AJUT directament sense comprovar els altres casos.

6.2.2 El personatge del jugador.

Tot i tenir una part del seu desenvolupament relacionada amb una **FSM** el seu funcionament a l'hora de fer les transicions entre estats no és tan directe com en el cas dels enemics. La seva diferencia més important respecte als enemics és que el jugador controla al personatge. Això fa que part de la seva implementació ha de tenir en compte aquest fet. Aquest problema és degut a que el joc fa un **polling** per comprovar si hi ha alguna tecla polsada que es correspongui a alguna acció assignada i llavors entra a l'estat corresponent i realitza aquesta acció. En cas contrari si no se li indica una altra cosa torna al seu estat per defecte. Això fa obligatori l'ús de variables de control per assegurar que els estats es mantinguin fins al final tant si son per part d'una acció del jugador com si és com a conseqüència d'una acció enemiga.

6.2.2.1 Funcionament general del jugador.

El primer pas de cada iteració del bucle consisteix en avaluar les variables de control.

- Mostra la interfície del jugador (salut, vides i puntuació).
- Controla que si està atacant i la animació corresponent a l'estat no ha arribat al final continuï realitzant l'atac.
- Si la salut li arriba a 0 o s'acaba el temps el jugador mor (passa a l'estat MUERTO).
- Controla la resta de variables de control (**atacado**, **cayendo** i **saltando**) per mantenir els seus corresponents estats (ATACADO,CAYENDO,SALTANDO).

DESPLAÇAMENT:

- Si es pressionen les fletxes direccionals del teclat (←,↑,→,↓) i no està bloquejant el personatge es desplaça (encara que es mantinguin polsades) per l'escenari tant si està a terra com si està saltant, però no pot desplaçar-se a través de les dureses de l'escenari.
- Si es desplaça cap a la dreta, el personatge es troba almenys a la meitat de la pantalla (de la finestra del joc, no del nivell) l'*scroll* el seguirà de manera que es manté al personatge centrar a partir d'aquest moment i fins que aquest arribi a un dels límits del nivell. Moment en que l'*scroll* s'atura fins nou avís.

SALT:

- Si el jugador prem la tecla de salt (D) i no està bloquejant, el personatge salta. Encara que es mantingui polsada només farà un salt i no podrà tornar a fer un altre salt fins que hagi aterrat (La animació finalitza). Això està controlat per la variable de control **mutexsaltando**.
- Passa a l'estat SALTANDO.

- El personatge realitza la animació corresponent al salt.

ATAC:

- Si es prem la tecla d'atacar (A) el personatge ataca però encara que es mantingui polsada no tornarà a atacar fins que hagi acabat l'atac actual. Si es manté polsada no pot tornar a atacar (ha de polsar per atacar). Aquest fet està controlat per la variable de control **mutexataque.**
- El jugador disposa de dos tipus d'atacs: un normal i un altre més fort que farà que l'enemic caigui a terra si no el bloqueja.
- El jugador realitza la animació corresponent a l'atac.
- Si l'atac col·lisiona amb un enemic aquest passa a canviar el seu estat a GOLPEADO i actua com s'ha explicat a l'apartat 6.2.2.1(funcionament general del jugador) .
- El personatge pot realitzar 2 atacs normals i seguidament un atac fort. Si passen 50 iteracions del bucle entre el primer atac i l'últim la combinació d'atacs torna a començar.

BLOQUEIG:

- Si es pressiona la tecla de bloqueig (S) i no està saltant passa a l'estat BLOQUEANDO.
- La variable de control **mutexbloqueando** controla quan està bloquejant i quan no.

6.2.2.2 Reaccions automàtiques del jugador.

Un cop explicat el funcionament del personatge en el qual hi intervé el jugador la resta són les reaccions y els seus estats corresponents a la **FSM**. A continuació s'expliquen aquests estats.

QUIETO:

- Estat per defecte del personatge. Únicament gestiona la animació corresponent a l'estat.

GOLPEADO:

- Gestiona la animació corresponent a l'estat.

SALTANDO:

- Gestiona la animació corresponent a l'estat i el seu desplaçament sense polsar tecles direccionals (Si es polsen aquestes tecles s'incrementa aquest desplaçament sempre que sigui possible).
- Controla que no hi hagi un canvi de sentit durant el salt. És a dir, començar un salt cap a un sentit concret (per exemple, cap a la dreta) i durant el salt canviar aquest sentir (per exemple canviar d'un salt a la dreta a un cap a l'esquerra durant el salt).

La figura 6.9 mostra al personatge fent un salt

Figura 6.9: Personatge saltant

BLOQUEANDO:

- Gestiona la animació corresponent a l'estat.
- Durant aquest estat no rep cap mal.

La figura 6.10 mostra al personatge del jugador bloquejant un atac.

Figura 6.10: Bloqueig d'un atac.

CAYENDO:

- Gestiona la animació corresponent a l'estat i el seu desplaçament (caiguda cap a la dreta o cap a l'esquerra segons l'impacte).
- Evita que el personatge es desplaci més enllà de la pantalla (controlat per *l'scroll*) o a través d'una duresa de l'escenari.

La figura 6.11 mostra al personatge caient per culpa d'un atac que el derriba.

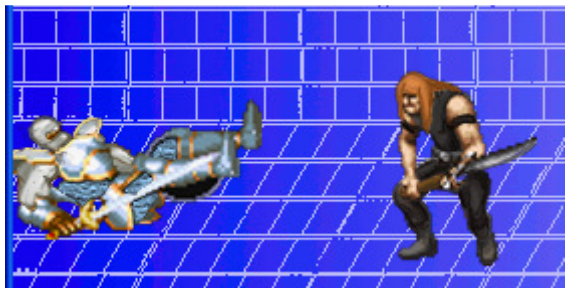


Figura 6.11: Caiguda del personatge

MUERTO:

- Similar a l'estat CAYENDO però un cop cau a terra perd una vida i recupera la seva salut. Si no li queden vides perd una continuació, recupera les vides y perd la meitat de la puntuació.
- S'inicialitza el temporitzador.
- Si no li queden vides ni continuacions crida al procés **gameOver(int fin)** i envia el senyal de fi del joc (game_over=1) que farà que el procés inicialitzi la interfície de fi del joc.

La figura 6.12 mostra la pèrdua de vides.

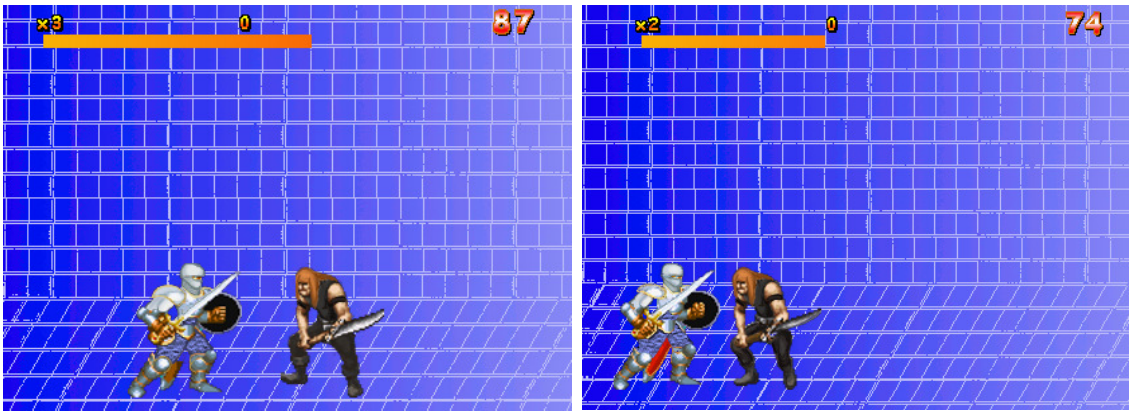


Figura 6.12: Pèrdua de vides.

6.2.3 Porta.

De tots els **agents** aquest és el més senzill ja que únicament comprova la dificultat y la posició del jugador.

6.2.3.1 FSM de la porta.

Té 3 estats tot i que l'últim no està desenvolupat explícitament ja que només fa referència a l'estat final en el que la porta es manté oberta. A continuació s'expliquen els altres dos estats. Tenint en compte que la dificultat ha de ser igual o superior a la Normal.

CERRADA:

- Si el jugador avança a la porta aquesta s'obre i gestiona la seva animació.
- Un cop oberta passa a l'estat ABIERTA.

ABIERTA:

- Fa que surti un enemic de la porta i incrementa el número d'enemics.
- Passa a l'estat FINAL el qual la manté oberta.

La figura 6.13 mostra el funcionament d'una porta. La primera part fa referència a la dificultat fàcil, de manera que no s'obre.

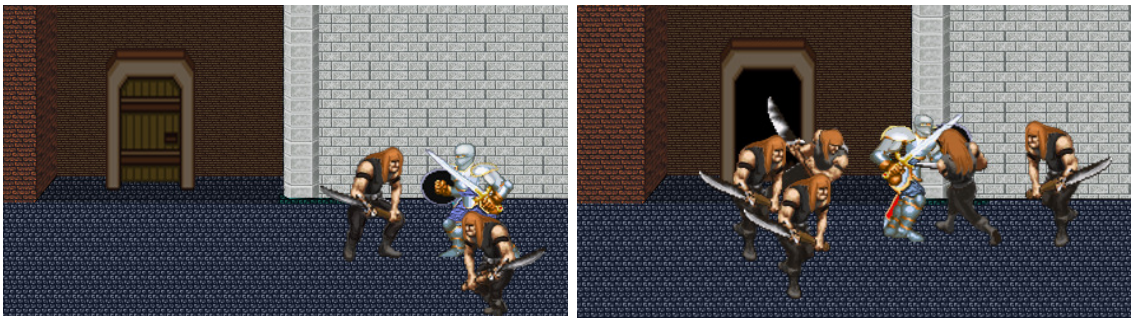


Figura 6.13: Funcionament de la porta.

6.3 Els escenaris.

Els escenaris formen el terreny pel qual tant el protagonista com els enemics es poden desplaçar (o no, si és el cas d'una duresa) i on es situen els objectes. **Bennu** incorpora un sistema per tractar les dureses basat en màscares de colors. Una màscara d'un escenari no és més que una plantilla del mateix mapa de l'escenari en 8 bits on el color negre simbolitza les superfícies sobre les quals es possible el desplaçament i la resta de colors simbolitzen diferents tipus de dureses (per exemple una paret, una plataforma o un forat) tot i que per a les dureses no hi ha cap color concret assignat (s'assignen al mateix programa). Aquest mateix sistema també l'utilitza la funció **path_find** pròpia del llenguatge per comprovar si hi ha algun camí.

La figura 6.14 mostra un escenari amb la seva corresponent màscara. Com es pot comprovar el negre fa referència al terreny transitable mentre que el blau indica que hi ha una paret o terreny impassable i el groc indica una porta de sortida que no es pot travessar per restriccions dels límits propis de l'escenari.

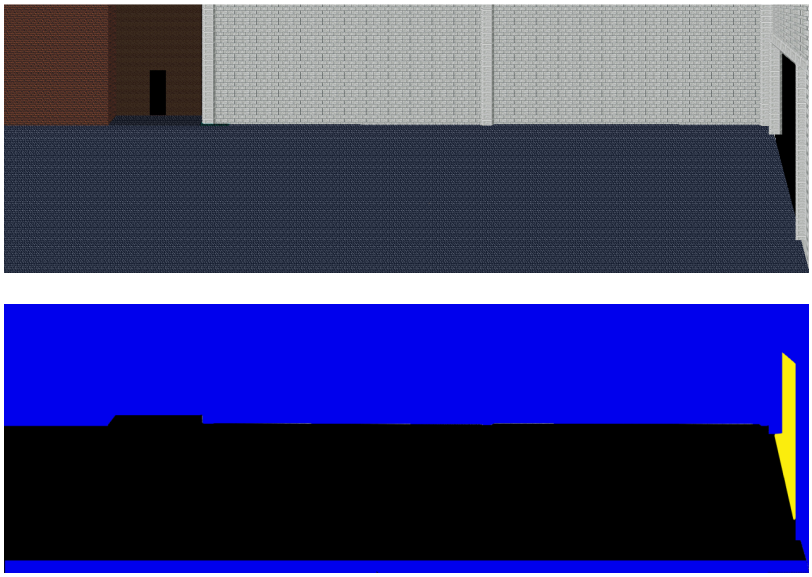


Figura 6.14: Escenari i la seva màscara

6.3.1 Gestió dels escenaris.

Els escenaris estan dividits en diverses seccions que s'han d'anar superant i cadascuna d'aquestes seccions està delimitada per diversos límits. La dificultat també influeix a l'hora de

limitar el màxim d'enemics que poden estar esperant (En cas d'una dificultat fàcil el límit és de 3 i a partir de la normal és 1). A continuació s'explica el funcionament de les diverses funcions relacionades amb els escenaris.

6.3.1.1 Funcions relacionades amb els escenaris.

Funció get_Stage(int stage_Actual):

- Comprova la dificultat i, segons la dificultat, actualitza el número màxim d'enemics que poden fer una determinada acció si cal.
- Carrega els *sprites* personatge del jugador i posteriorment crida al procés que s'encarrega de cada escenari (assignat pel paràmetre **stage_Actual**).
- Manté l'escenari desactivat fins que el procés **init_Stage(int stage)** l'activa.

Procés init_Stage(int stage):

Aquest procés s'encarrega d'assegurar el temps necessari per carregar tot el que calgui, a més d'aportar un detall estètic.

- Un text informant del número de pantalla passa per la interfície i s'espera uns segons abans d'enviar el senyal per activar la pantalla.
- Crida al temporitzador del nivell.

La figura 6.15 mostra aquest text introductori.

Figura 6.15: Funcionament del procés init_Stage amb el text informatiu.

Procés get_limites(int izquierdo, int derecho,int stage,int numlimite):

- Gestiona i manté en tot moment (per les comprovacions constants) els límits actuals del nivell corresponent.

Procés flechaGo():

- Mostra el gràfic parpellejant (3 cops) d'una fletxa que indica al jugador que pot continuar ja que ha superat un límit parcial del nivell.

La figura 6.16 mostra aquest senyal.

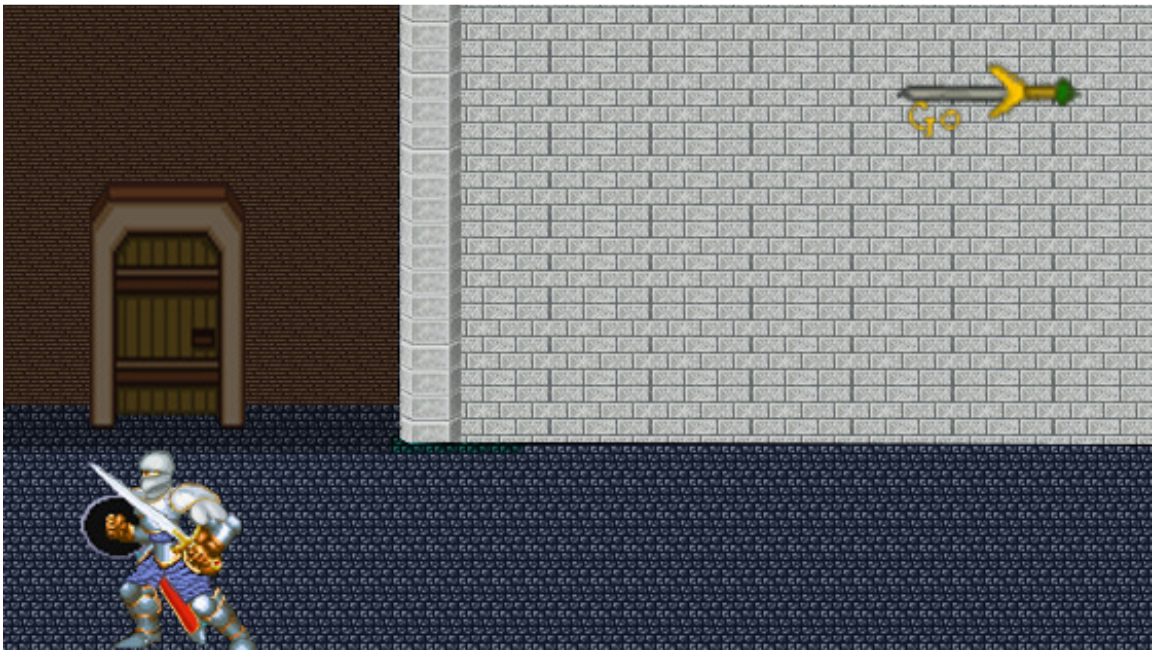


Figura 6.16: Indicador per poder continuar.

6.3.1.2 Escenari principal.

Aquest és l'escenari on transcorre l'acció del joc. A continuació s'explica el seu funcionament (funciona igual per a altres escenaris):

- Inicialitza les dureses i carrega els *sprites* dels enemics i la porta.
- Crida al protagonista i a la porta i els situa a l'escenari.
- Inicialitza l'*scroll* amb el seu mapa corresponent.
- Carrega els límits corresponents a l'escenari amb el procés **get_limites(int izquierdo, int derecho,int stage,int numlimite)**.
- Carrega els enemics (en funció de la dificultat) corresponents al primer límit incrementant el seu corresponent comptador (per saber quants enemics queden).
- Si el comptador d'enemics arriba a 0 (perquè moren o fugen) s'avisa al jugador amb el procés **flechaGo()** i es carreguen els enemics del següent límit.
- Si s'han superat tots els límits es calcula la puntuació total amb la funció **puntuacion_total()**, es guarden les vides i la puntuació i es carrega el nivell següent.

6.4 Altres funcions i processos.

Procés Main():

- Inicialitza la interfície general (resolució, mode de finestra i profunditat de color).
- Carrega els mapes, les màscares corresponents als mapes i les fonts necessàries.
- Inicialitza vides i continuacions i el primer nivell.
- Crida al primer nivell del joc.
- Si rep el senyal per canviar de nivell modifica la dificultat amb la funció **cambia_dificultat()** i carrega el nivell següent.
- Si es prem la tecla 'Esc' acaba el joc.

Procés animar (int primero , int ultimo):

- Comprova que el seu pare (procés que el crida) encara existeix (és viu) i gestiona la animació de l'estat corresponent a partir dels paràmetres rebuts.

Procés tiempo():

Temporitzador de cada límit del nivell.

- Inicialitza el temporitzador i el mostra a la pantalla.
- Cada segon aproximadament es decrementa.
- Si arriba a 0 envia senyal de **time_up** y el jugador mor.

Procés gameOver(int fin):

- Si rep el senyal de fi del joc desapareix tot el que hi ha a la pantalla i s'elimina la resta de processos actius.
- Mostra el text "GAME OVER".
- Si es prem la tecla 'Esc' acaba el joc.

Funció get_decision(int max):

- Genera un número aleatori entre 0 i el paràmetre rebut.

Funció set_estadoProximaPosicion(int factor, int procesoID):

Funció que s'encarrega de planificar l'estratègia dels enemics de forma similar a la que es descriu a l'apartat 5.2.2.2 del capítol anterior.

- Aquesta funció contempla tant el cas de la dificultat Fàcil com la resta de dificultats.

Funció set_estadoAproximacion(int objetivoID, int procesoID):

- Aquesta funció s'encarrega de fer que l'enemic s'acosti caminant o bé corrents cap al jugador si hi ha suficient distància com per a poder córrer i atacar (1/3 de la resolució corresponent a l'eix x).

Funció get_destino_aprox(int coord, string tmpestado,int IDobjetivo):

- Aquesta funció calcula una posició aleatòria per a que l'enemic pugui esperar o vigilar al jugador.

Funció get_destino_apoyo(int coord, string xoy):

- Calcula una posició de destí per a l'enemic de manera que quedi a prop del jugador però sense atacar.
- Cal distingir entre si és una posició lateral (x) o vertical (y). Això es fa mitjançant el paràmetre **xoy**.

Funció cambio_y(int Mapa_durezas,int Durezas,int IDx):

- Canvia la posició y de l'enemic un cop aquest surt de la posició de la càmera (surt de la finestra del joc).

Funció get_destinoyHuida(int Mapa_durezas,int Durezas,int destinoX):

- Indica a l'enemic que estigui fugint una alçada de destí, evitant les dureses.

Funció avisaProc(int IDexcluida):

- Comprova entre tots els enemics que queden i no estan fugint quin és el que té més salut.
- Un cop tria a un enemic li canvia l'estat a APROX_JUGADOR.

Funció puntuacion_total():

- Calcula una bonificació de temps a través del temps restant.
- Calcula un bonificació per la dificultat.
- Incrementa la puntuació del jugador sumant aquestes dues bonificacions.

Funció cambia_dificultat():

- Estableix un rang (varia segons el nivell jugat i la dificultat) per avaluar la puntuació segons la dificultat.
- Si la puntuació està per sobre del valor màxim d'aquest rang s'incrementa la dificultat.
- Si la puntuació està per sota del valor mínim d'aquest rang disminueix la dificultat.

Procés energia_jugador(int barra):

- Crea la barra de salut del jugador i la situa a la interfície.

7. Test i proves

Les proves que s'han realitzat per verificar el funcionament correcte del joc es divideixen en proves **estètiques** i **tests funcionals**. Les proves estètiques impliquen les animacions i la correcta visualització tant dels escenaris com de la interfície o els personatges (jugador i enemics). Per altra banda els tests funcionals es centren en el correcte funcionament de les **Màquines d'estats finits**, el comportament dels agents, el bon funcionament de les funcions i altres processos, la gestió dels escenaris i, en general, tot el que fa referència a l'execució del joc i correcció de *bugs*[21][RB14].

7.1 Escenari de Test.

Per realitzar la majoria de les proves, tant estètiques com funcionals, s'ha creat un escenari especial com el que es mostra a la figura 7.1

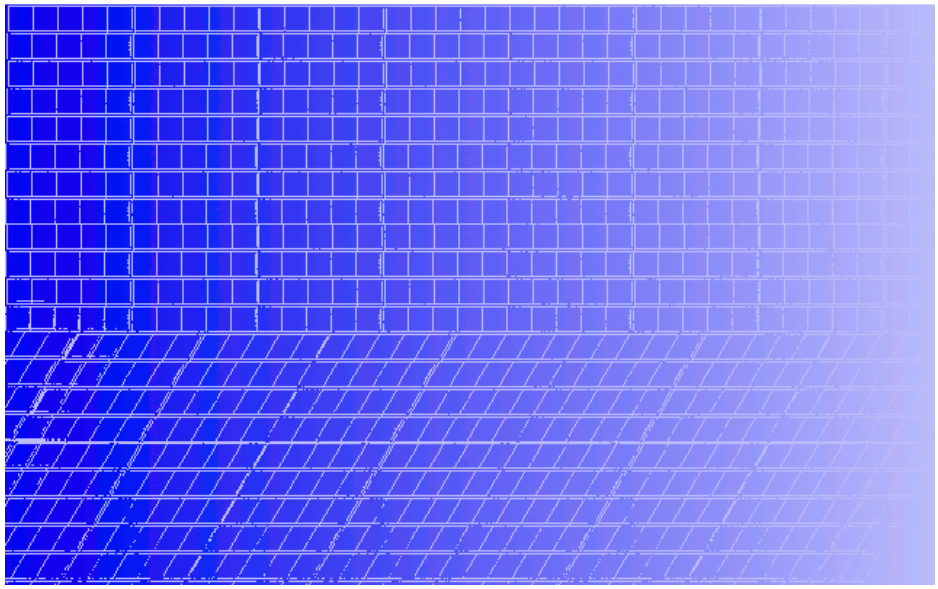


Figura 7.1: Escenari de Tests.

Com es pot veure tant la paret com el terra tenen cel·les mesurades de manera que es poden fer servir com a referència a l'hora de mesurar i comprovar les posicions dels personatges durant les animacions i fer les diferents proves que requereixin certa precisió (per exemple el moviment d'*scroll*, col·lisions entre personatges o amb l'escenari i el posicionament).

7.2 Proves de les animacions.

Les proves de les animacions consisteixen principalment en verificar els *sprites* corresponents a cada animació i en verificar el posicionament correcte dels *sprites* individualment.

Els *sprites* s'han de comprovar tant a nivell de qualitat de la imatge (fons transparent, sense píxels buits i un mínim de detall i definició) com a nivell de la visualització de la animació

[21] Es tracta del resultat d'un error de programació en el procés de creació de software. Aquest error pot presentar-se en qualsevol de les etapes del cicle de vida del software, tot i que els més evidents sorgeixen durant la etapa de desenvolupament i programació.

(correlació entre imatges). El propi **FPGEdit** incorpora opcions per comprovar aquests dos aspectes.

Pel que fa al posicionament dels *sprites* que formen part d'una animació, cal tenir en compte que el llenguatge **Bennu** afegeix a les imatges punts de control que es poden fer servir tant per al posicionament de la imatge al joc com per adherir altres *sprites* (per exemple armes) a l'*sprite* principal. Generalment **Bennu** posiciona els *sprites* segons el seu centre geomètric (píxel central). De manera que una animació amb *sprites* de diferents proporcions pot modificar la seva posició automàticament (cosa que s'ha d'evitar). Per fer això el llenguatge conté una funció que permet canviar de posició aquest centre. Per calibrar els centres dels *sprites* i aconseguir que el personatge es mantingui al mateix punt s'ha creat un programa que utilitzant l'escenari de Test del punt 7.1 carrega un fitxer .fpg amb els *sprites* del personatge que es vol tractar permet triar una animació concreta (indicant el seus *sprites* inicial i final corresponents) i cada cop que es prem la tecla 'A' l'animació passa al seu *sprite* següent. D'aquesta manera es pot comprovar imatge per imatge si cal modificar el seu centre amb la funció **set_center(int FileID, int GraphID , int x, int y)**[Annex 2].

7.3 Tests funcionals.

Les proves funcionals s'han realitzat de manera progressiva a mesura que s'han anat afegit funcionalitats al joc o estats a les **FSM**. Les proves s'han realitzat, per una banda, a temps real (al joc directament) per comprovar el seu correcte funcionament global i que hi hagi una correcta interacció amb tots els elements que formen part el joc. Per l'altra banda, aprofitant les pròpies eines del compilador que proporciona el propi llenguatge.

7.3.1 Proves a temps real.

Aquest tipus de proves s'han fet per comprovar els errors d'execució amb tots els elements del joc en funcionament i per comprovar la interacció dels enemics amb el jugador.

També s'han aprofitat diverses funcions de tipus **write()** o **write_var()**[Annex 3] per mostrar en tot moment valors que facin referència als processos dels **agents** (ID dels processos, posició, distància entre dos processos o canvis d'estats a la **FSM**).

7.3.2 Eines del compilador.

El propi compilador de **Bennu** incorpora eines que permeten depurar el programa. Les més usuals son la funció **say()**[Annex 4], que permet mostrar per la consola del compilador un missatge, una variable o el retorn d'una funció o el propi depurador del compilador que permet realitzar diverses operacions per comprovar el funcionament del programa.

7.3.2.1 Depurador.

Per poder fer servir el depurador cal fer 3 passos previs:

- Incloure el mòdul "mod_debug".
- Compilar en mode de depuració. D'aquesta manera s'inclou informació addicional (per exemple: noms de variables o posicions de les instruccions al codi font) a l'arxiu compilat.

- Durant el joc es pot accedir al depurador pressionant ALT+C. D'aquesta manera el joc queda congelat i apareix una altra consola des de on es poden executar les comandes de depuració. La taula 7.1 mostra aquestes comandes i el seu funcionament.

Taula 7.1 Comandes de la consola de depuració.

Variables[Annex 5]	
GLOBALS	Mostra totes les variables globals (inclús les predefinides pel llenguatge) i els seus valors actuals.
LOCALS	Mostra el tipus, nom i valors de totes les variables locals.
PUBLICS nomProcés	Mostra el tipus, nom i valors de totes les variables públiques d'un procés.
PRIVATES nomProcés	Mostra el tipus, nom i valors de totes les variables privades d'un procés.
nomProcés.nomVariable	Mostra el valor d'una variable local, pública o privada específica per a un procés concret.
Processos	
INSTANCES	Mostra un llistat dels identificadors de procés de totes les instàncies existents a l'instant actual indicant, a més, el seu estat (actiu, sleep, freeze, mort o a punt de morir).
RUN nomProcés [parametres opcionals]	Executa una nova instància del procés indicat
KILL identificadorProcés	Elimina el procés indicat.
WAKEUP identificadorProcés	Desperta el procés indicat.
SLEEP identificadorProcés	Adorm un procés.
FREEZE identificadorProcés	Congela un procés.
KILLALL nomProcés	Mata totes les instàncies del procés indicat.
WAKEUPALL nomProcés	Desperta totes les instàncies del procés indicat.
SLEEPALL nomProcés	Adorm totes les instàncies del procés indicat.
FREEZEALL nomProcés	Congela totes les instàncies del procés indicat.
Breakpoints	
BREAK	Mostra un llistat dels breakpoints.
BREAK nomProcés	Crea un breakpoint a la execució del procés indicat.
BREAKALL	Agrega un breakpoint a totes les instàncies de tots els processos.
BREAKALLTYPES	Agrega un breakpoint a tots els tipus de processos.
DELETE nomProcés	Elimina el breakpoint del procés
DELETEALL	Elimina tots els breakpoints de totes les instàncies.
DELETEALLTYPES	Elimina tots els breakpoints de tots els tipus de processos.
Altres comandes	
QUIT	Surt de la consola de depuració i finalitza el programa.
SHOW expressió	Realitza una expressió aritmètica i mostra el resultat.
VARS	Mostra les variables i el valor de la consola de depuració.
FILES	Variable interna de la consola que mostra arxius oberts.
STRINGS	Mostra totes les cadenes existents a memòria.

Fent ús d'aquestes comandes i variables es pot fer que un determinat procés realitzi unes accions determinades. Per exemple, en el cas d'un enemic es poden modificar els seus estats de manera que facin les accions desitjades.

7.4 Proves de compatibilitat.

El software s'ha comprovat en diferents versions de Windows (XP, Vista, 7) sense canvis aparents en el seu rendiment i a la execució.

8. Conclusions finals

8.1 Realització dels objectius.

L'objectiu principal del projecte era el de desenvolupar un *engine* basat en els videojocs que durant els anys 90 (i encara avui) tenien un gran èxit. A més a aquest *engine* se li ha afegit un pes important a la **intel·ligència artificial**.

S'ha creat un joc tipus *beat'em* up en 2D i escenaris amb profunditat de moviment, de manera que es garanteix una gran quantitat de possibilitats de moviments i d'accions per part de tots els personatges assegurant, a més, que es consumeixin pocs recursos.

Pel desenvolupament de la **I.A.** s'han utilitzat **Màquines d'Estats Finits** on els seus estats defineixen les accions (voluntàries o involuntàries) que els diferents **agents** poden realitzar. D'aquesta manera es poden identificar i programar les seves accions tant voluntàries com involuntàries de manera independent. També hi ha una part important destinada al tractament de la **concurrència** entre els processos. Per una banda la **concurrència** s'ha utilitzat per evitar situacions conflictives entre els processos dels **agents** enemics i també s'ha fet servir per establir un sistema de comunicació (mitjançant la sincronització entre processos) també entre els enemics ja que també s'ha programat l'**entorn multiagent** que fa que els enemics es coordinin per atacar o realitzar estratègies d'atac.

Les accions voluntàries dels enemics son: atacar directament, bloquejar un atac del jugador, planificar estratègies o, si estan ferits poden arribar a fugir. Per altra banda les accions involuntàries dels enemics son: rebre un atac del jugador, caure o morir.

Les tecles polsades (tant per les accions del jugador com per altes opcions com ara finalitzar el joc) es controlen mitjançant **polling**.

Un altre fet destacat del joc és que la seva dificultat varia segons la habilitat del jugador, de manera que s'adapta (augmentant o disminuint) al seu progrés. La dificultat influeix en les accions que poden realitzar els enemics, en les seves decisions, en el número d'enemics que el jugador haurà de derrotar abans que s'acabi el temps i en la interacció d'altres elements (enemics ocults).

Els escenaris dels nivells estan dividits en seccions que el jugador ha d'anar superant (eliminant els enemics) abans que s'acabi el temps. Per fer-ho s'ha implementat un temporitzador que es reinicia cada cop que es supera una secció del nivell o el jugador mor.

El llenguatge triat per al seu desenvolupament ha sigut **Bennu**, ja que està especialment enfocat al desenvolupament de videojocs i permet una gran portabilitat. Aquest fet ha implicat un procés previ d'aprenentatge del llenguatge i les particularitats que aquest té, com ara la programació orientada als processos.

Un cop finalitzat el treball, s'ha provat i verificat el seu correcte funcionament. S'ha analitzat especialment la jugabilitat i la motivació que proporciona al jugador, ja que és l'usuari a qui el joc va destinat.

8.2 Valoració personal

Durant gairebé tot el procés de desenvolupament del projecte (exceptuant la part corresponent a l'aprenentatge del llenguatge) s'han aplicat una gran quantitat dels coneixements, recursos i mètodes de treball apresos durant el transcurs de la carrera universitària.

Aquest projecte, tot i ser laboriós, pot servir com una aplicació diferent dels recursos assumits ja que implica conceptes com la **creació d'algorismes**, **tècniques d'intel·ligència artificial** (**Màquines d'estats finits** i **comunicació entre agents** aprofitant tècniques de sincronització entre processos) i **control de processos** (concurrència i operacions entre processos).

9. Línies d'ampliació futures

El projecte pot incloure diverses ampliacions. De fet tantes com es vulguin realitzar ja que es pot considerar perfectament la base per un joc complet, amb els elements indispensables. A continuació es proposen algunes de les més habituals.

9.1 Nous enemics.

Un sol tipus d'enemic pot ser poc per a un joc complet, així doncs es poden afegir nous tipus d'enemics. Poden ser enemics que ataquin a distància, enemics amb una **I.A.** més desenvolupada que altres i fins i tot enemics de final de nivell. Fins i tot es podrien afegir variants entre els mencionats anteriorment i el que ja hi ha creat (més força, menys defenses o altres *sprites*).

Tots aquests enemics tindrien les seves corresponents **Màquines d'estats finits** que poden ser similars entre elles, en el cas d'enemics semblants o bé amb estats i transicions pròpies si son enemics diferents i tots es podrien sincronitzar entre ells.

9.2 Múltiples nivells diferents.

Es poden crear nous escenaris que es vagin alternant a mesura que el jugador va avançant. Això permetria poder crear una *storyline* per al joc i fer-lo més atractiu. També es poden afegir camins separats als escenaris de manera que no hi hagi un únic camí per arribar al final i superar el joc. Aquests camins alternatius es poden aconseguir aprofitant el control de les dureses dels escenaris junt amb el sistema de seccions dels nivells.

9.3 Multijugador.

Es pot afegir la opció per a un segon jugador (local o per xarxa) que tindria un altre personatge. Seria interessant afegir la possibilitat de triar entre diversos personatges. Únicament caldria modificar els *sprites* per al personatge.

El multijugador podria ser durant el joc principal o bé un mode 1 contra 1 en el qual un jugador hauria de derrotar a l'altre com si es tractés d'un enemic.

9.4 Opcions.

Es pot afegir una pantalla principal amb un menú d'opcions que permeti modificar paràmetres com la dificultat, la resolució de pantalla o el teclat.

9.5 Guardar dades.

Sempre és atractiu el fet de poder comparar el progrés amb un mateix o amb altres. Per això es pot implementar un sistema per guardar els resultats finals dins d'un *ranking.*

Si el joc resulta ser suficientment llarg es pot afegir la opció de salvar la partida.

9.6 Realitzar un port per consoles.

Aprofitant la compatibilitat que té el propi llenguatge es pot realitzar un port per a consoles de videojocs. Poden tractar-se de ports oficials com ara la **GP2X** o bé altres que puguin requerir

certes modificacions al codi. En qualsevol cas cal fer un *mapeig* (assignació) dels botons dels corresponents comandaments per poder ser interpretats.

9.7 Afegir altres elements o objectius.

Un aspecte que pot ser atractiu és el fet de poder aconseguir objectes que puguin potenciar al jugador ja sigui amb punts, salut o vides. També es podrien incloure nous objectius com ara haver d'aconseguir un determinat objecte per poder continuar o arribar a un punt concret encara que hi hagi enemics supervivents.

10. Bibliografia

10.1 Llibres

- [RB1] Brian Schwab, AI Game Engine Programming. Charles River Media, 2004.
- [RB2] Stuart J. Russell i Peter Norving, Inteligencia Artificial. Un Enfoque Moderno. Pearson Prentices Hall, 2006.
- [RB3] Jesús Carretero Pérez, Félix García Carballeira, Pedro de Miguel Anasagasti i [RB4] Fernando Pérez Costoya, Sistemas Operativos. Una visión aplicada. Mc Graw Hill, 2001.
- [RB5] William Stallings, Sistemas Operativos. Aspectos internos y principios de diseño. [RB6] Pearson Prentices Hall, 2005.
- [RB7] Quim Borges, Joan Serra i Josep M. Arqués, Teoria d'autòmats. Universitat Autònoma de Barcelona. Servei de Publicacions, 1996.

10.2 Documentació online.

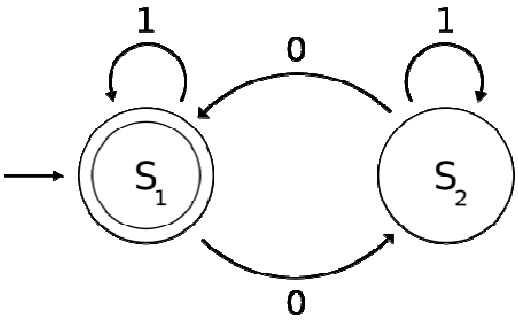
- [RB8] [http://trinit.es/tutoriales] Tutorial de BennuGD (19-9-2010).
- [RB9] [www.bennugd.org/downloads/ManualBennuGD_Osk.pdf] Curso de iniciación a la programación de videojuegos con el lenguaje Bennu v1.0 (en Windows y GNU/Linux) (Sense Data de publicació).
- [RB10] [http://wiki.bennugd.org] Documentació de Bennu (16-2-2010).
- [RB11] [http://es.wikipedia.org/wiki/Problema_del_barbero_durmiente] Problema del barbero durmiente - Wikipedia, La enciclopedia libre (1-6-2011).
- [RB12] [http://es.wikipedia.org/wiki/Beat%27em-up] *Beat'em up* - Wikipedia, La enciclopedia libre (11-6-2011).
- [RB13] [http://es.wikipedia.org/wiki/Engine] Motor de videojuego - Wikipedia, La enciclopedia libre (12-6-2011).
- [RB14] [http://es.wikipedia.org/wiki/Bug] Error de software - Wikipedia, La enciclopedia libre (1-6-2011).
- [RB15] [www.bennugd.org] Bennu Game Development (20-6-2011).
- [RB16] [http://bennupack.blogspot.com] Bennupack (18-3-2011).
- [RB17][http://vidaartificial.com/index.php?title=Maquinas_de_Estados_Finitos_(ai-epot.com)] Maquinas de Estados Finitos (ai-depot.com) (27-5-2006).
- [RB18] [http://en.wikipedia.org/wiki/List_of_game_engines] List of game engines – Wikipedia, The free Encyplopedia (20-6-2011).
- [RB19] [http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones] Interfaz de programación de aplicaciones – Wikipedia, La enciclopedia libre (15-6-2011).
- [RB20] [http://es.wikipedia.org/wiki/Homebrew] Homebrew - Wikipedia, La enciclopedia libre (24-6-2011).
- [RB 21] [http://es.wikipedia.org/wiki/Script] Script – Wikipedia, La enciclopedia libre (7-5-2011).
- [RB22] [http://es.wikipedia.org/wiki/Adobe_Flash] Adobe Flash Professional – Wikipedia, la enciclopedia libre (9-6-2011).
- [RB23] [http://es.wikipedia.org/wiki/Adobe_Flex] Adobe Flex – Wikipedia, la enciclopedia libre (23-6-2011).

- [RB24] [http://es.wikipedia.org/wiki/DirectX] DirectX – Wikipedia, la enciclopedia libre (13-6-2011).
- [RB25] [http://es.wikipedia.org/wiki/OpenGL] OpenGL – Wikipedia, la enciclopedia libre (25-5-2011).
- [RB27] [http://es.wikipedia.org/wiki/Switch_case] Switch case – Wikipedia, la enciclopedia libre (26-5-2011).
- [RB28] [http://es.wikipedia.org/wiki/Polling] Polling - Wikipedia, la enciclopedia libre (12-10-2011).

Annex

Annex 1: Esquema d'una Màquina de Moore.

La figura A2 mostra un esquema senzill d'una **Màquina de Moore** a on S_1 i S_2 són els estats.



A1: Esquema senzill d'una Màquina de Moore.

Annex 2: Funció set_center(int FileID, int GraphID , int x, int y).

Permet canviar el centre d'un gràfic, especificant les noves coordenades per al mateix. Aquestes coordenades sempre estan dins del gràfic i el punt (0,0) es correspon amb el cantó superior esquerre i les coordenades creixen cap avall i cap a la dreta.

Paràmetres:

- int FileID: Número de llibreria FPG.
- int GraphID: Número del gràfic dins de la llibreria.
- int X: Nou valor per a la coordenada horitzontal del centre.
- int Y: Nou valor per a la coordenada vertical del centre.

La figura A2 mostra un exemple de codi:

```
//Modifica el centre de les imatges para que no s'alteri la seva posició
        set_center(file1,14,65,57);
        set_center(file1,15,42,76);
        set_center(file1,16,46,56);
        set_center(file1,17,51,42);
```

Figura A2: Exemple de codi de la funció set_center.

Annex 3: Algunes Funcions per mostrar text per pantalla.

write():

Mostra qualsevol text fix per pantalla. Pot ser un text normal, el valor d'una variable. Rep una sèrie de paràmetres que indiquen les característiques d'aquest text i el text.

La seva definició és:

int write(integer, integer, integer, integer, string)

Paràmetres:

1. Codi corresponent a l'estil de lletra. En cas d'utilitzar l'estil per defecte el valor és 0.
2. Coordenada horitzontal en píxels.
3. Coordenada vertical en píxels.
4. Codi corresponent a la alineació del text respecte a la posició X,Y indicada. Pot ser indicat per un valor numèric amb un valor del 0 al 8 o bé amb les constants predefinides del propi llenguatge com es mostra a la taula A3.

   Taula A3: Constants de per a la alineació del text.

0	**ALIGN_TOP_LEFT**
1	**ALIGN_TOP**
2	**ALIGN_TOP_RIGHT**
3	**ALIGN_CENTER_LEFT**
4	**ALIGN_CENTER**
5	**ALIGN_CENTER_RIGHT**
6	**ALIGN_BOTTOM_LEFT**
7	**ALIGN_BOTTOM**
8	**ALIGN_BOTTOM_RIGHT**

5. Text a mostrar.

write_var():

Mostra el resultat d'una variable indicada. A diferència de la funció write(), aquesta no és mostra el resultat en un text fix, de manera que si el resultat de la variable canvia, el text mostrat s'actualitza tot sol.

La seva definició és:

int write_var(integer, integer, integer, integer, variable)

On els paràmetres son els mateixos que per a la funció write() excepte l'últim, que és la variable de la que es vol mostrar el valor.

Annex 4: Funció say().

Mostra un text (passat per paràmetre) per la consola del compilador. Aquest text pot ser un *string* o bé el resultat d'una variable.

Annex 5: Àmbit de variables i constants del llenguatge.

Bennu disposa de diferents àmbits per a las variables. Alguns són freqüents en altres llenguatges mentre que d'altres son propis d'aquest llenguatge.

- **Globals:** S'hi poden accedir des de qualsevol punt del programa tant per verificar el seu valor com per modificar-lo. Es declaren abans de qualsevol procés o funció (inclòs el procés principal) i dins del seu bloc corresponent "GLOBAL/END".
- **Locals:** Existeixen a cada procés però cadascun te la seva variable pròpia. Tenen el mateix nom però el valor és independent al dels altres processos. Qualsevol procés o funció pot accedir a les variables locals d'altres processos (a través de la ID del procés) per obtenir el valor o modificar-lo. Es declaren abans de qualsevol altre procés dins del seu bloc corresponent "LOCAL/END".

- **Privades**: Variables d'àmbit exclusiu al procés o funció que les crea. Es declaren dins del bloc "PRIVATE/END" corresponent al procés o funció.
- **Constants:** No son variables pròpiament dites ja que el seu valor no es pot modificar i no s'ha d'especificar el tipus de dada que emmagatzemen. De la mateixa manera que les globals el seu àmbit és de tot el programa. Es declaren abans dels blocs "GLOBAL/END" i "LOCAL/END" dins del bloc "CONST/END".

Signatura:

L' autor: Carlos Adan López