Treball Final de Carrera

Desenvolupament d'un joc per torns parametritzable amb possibilitat multijugador i I.A.

Joan Casas Roma

Enginyeria Tècnica d'Informàtica en Informàtica de Gestió

Directors: Jaume Vila Serra i Jordi Surinyac Albareda

Vic, juny de 2008

A tots aquell qui, durant aquest final de curs, m’han aguantat en el moments més insuportables, fins i tot quan ni jo mateix m’aguantava.

Índex

Universitat de Vic
Escola Politècnica Superior

Resum de Trebal Final de Carrera

Enginyeria Tècnica d'Informàtica de Gestió

Títol: Desenvolupament d'un joc per torns parametritzable amb possibilitat multijugador i I.A.

Paraules clau: Joc, 2D, DirectX

Autor: Joan Casas Roma

Direcció: Jaume Vila Serra i Jordi Surinyac Albareda

Data: Juny de 2008

Resum

Els jocs són una de les indústries de software més gran del món de la informàtica. Des dels primers jocs en blanc i negre que simulaven raquetes i una pilota, fins avui en dia, en que el desenvolupament d'un joc porta darrere un equip de professionals tant o més gran que el major dels projectes informàtics del món de les indústries, els jocs han evolucionat més que la majoria de programes.

La possibilitat d'elaborar un joc és, a part d'una proposta temptadora (ja que difereix enormement de qualsevol pràctica feta durant la carrera), un repte de caire personal per algú que sempre ha estat en contacte amb videojocs i que, després d'adquirir una sèrie de coneixements indispensables, s'ha proposat d'intentar desenvolupar-ne un des de l'arrel. L'objectiu d'aquest treball és doncs això, aprendre com neix un joc partint de res, i veure totes les complicacions que sorgeixen a l'hora de desenvolupar-lo.

Els resultats delt projecte mostren generosament el gran nombre de problemes que sorgeixen en un procés com aquest, però com a conclusions importants cal destacar la satisfacció envers els resultats obtinguts, així com els coneixements que s'han guanyat mitjançant el desenvolupament del programa.

Universitat de Vic
Escola Politècnica Superior

TFC summary

Technical Computing Engineering

Title: Turn-based strategy game developement with multiplayer possibility and A.I.

Keywords: Game, 2D, DirectX

Author: Joan Casas Roma

Management: Jaume Vila Serra i Jordi Surinyac Albareda

Date: June 2008

Summary

Videogame is one of the greatest software industries in the world of computer science. From the first black and white games that simulated rackets and a ball, until nowadays, where the development of a game needs a team of professionals as big or even bigger than the biggest of the computer projects in the industry world, videogames have evolved more than the majority of programs.

The chance of elaborating a game is not only a very temptative proposal (because it hugely differs from any work done during the degree), but also a personal challenge for someone that has always been in touch with this kind of games and who, after acquiring a bunch of absolutely indispensable concepts, has decided trying to develop a game from the root. This is the goal of this work, to learn how a game is created from nothing, and to face all the complications that appear while developing it.

The results of the project show widely the big amount of problems that appear during a process such as this, but concluding I must highlight the satisfaction of what I have achieved, as well as the knowledge acquired through the development of the program.

1. Introducció

Tothom, qui més qui menys, ha tingut alguna vegada relació amb el món del videojocs. Des dels seus inicis fins avui en dia han patit un canvi espectacular que, per si fós poc amb el seu aspecte, es fa palès amb tot l'equip de professionals que hi ha actualment al darrere de qualsevol projecte en aquest món.

Tenint en compte això, i sabent que és molt pretensiós intentar fer un joc equiparable a qualsevol dels existents al mercat avui en dia, sí que la curiositat per aprendre què s'amaga darrere d'aquests, i com funciona el procés de creació d'un joc des dels seus inicis, han fet que un servidor decidís emprendre aquest projecte. Partint d'un projecte qualsevol en blanc, i un paquet de llibreries gràfiques totalment desconegudes, aquest projecte ha suposat un continu descobriment de les possibilitats que un llenguatge actual i un paquet de llibreries poden oferir a un programador que ni tant sols ha acabat la carrera (a part de tot això, aquest projecte hauria de poder ser el visat per tal cosa). A més de tot el coneixement derivat de les aplicacions gràfiques i sonores del projecte, el seu enfocament a objectes ha fet que augmentés enormement el meu coneixement sobre aquests temes (de gran importància en el món de la programació).

Així doncs, aquest és un projecte que comença amb una idea que de ben segur molta gent a molta gent l'hi ha passat pel cap, que s'embarca en un terreny completament desconegut inicialment per un servidor, i que finalitza amb un resultat molt satisfactori tant a nivell de programa com sobretot a nivell d'aprenentatge i formació com a programador.

2. Objectius del projecte

Crear un joc d'estratègia per torns en el qual el jugador disposarà d'un avatar (un personatge a qui controlarà, veure Glossari) que podrà moure a través d'un mapa i que podrà fer interactuar amb diferents elements del mapa, així com un edifici (per exemple un castell) on el jugador podrà construïr estructures que li permetran després reclutar criatures que acompanyaran l'avatar.

L'apartat gràfic del joc funcionarà a través d'un sistema de *tiles* i *sprites* en 2D (s'entén per un *tile* una imatge bidimensional estàtica que es pot dibuixar de forma contígua sense presentar anomalíes en el terreny que representa, i per un *sprite* una imatge estàtica també bidimensional que representa per exemple una criatura o un avatar, podent ésser aquest completament estàtic o animat a través d'altres *sprites* semblants que representin les diferents fases del moviment). Tant uns com els altres es trobaran emmagatzemats en uns fitxers d'imatge. La mida dels *sprites* serà variable en funció de què representin, però la mida dels *tiles* serà sempre de 32 x 32 pixels (per tal de poder dividir la superfície del mapa en una cuadrícula sense problemes de *tiles* de diferents mides).

Aquests gràfics (tant *tiles* com *sprites*) es podràn parametritzar a través d'uns fitxers de text des d'on el joc agafarà els arxius que contenen els gràfics. Canviant el nom dels fitxers d'imatge en els corresponents fitxers de text, per tant, farà que el programa utilitzi aquests nous fitxers d'imatge per obtenir els gràfics que utilitzarà. A part d'aquests dos tipus de fitxers (que contindran *tiles* o *sprites*) es podran parametritzar també fitxers d'imatge estàtics, com per exemple els fons de les batalles, o el fons de la imatge del castell. Així mateix també serà possible parametritzar la música, que es llegirà de fitxers *.mp3*. Això farà doncs que pel que fa a l'aspecte del joc es pugui variar àmpliament en funció de les fonts gràfiques i sonores que s'utilitzin.

Pel que fa al funcionament del joc en sí, i malgrat ja se n'ha comentat la base (deixant a part que existeix una secció en aquesta memòria que explica amb més detall el funcionament de cara a l'usuari del joc), hi haurà tres modes de joc disponibles:

- Monojugador (o mode història): Un sol jugador es desenvolupa per un mapa sense cap oponent del seu mateix nivell, sinó sols

amb oponents estàtics que formen part del mateix mapa. La idea és que aquest mode consta d'un seguit de mapes encadenats que representen una història i que la superació d'un permet l'accés a l'altre. L'objectiu d'aquests mapes serà el mateix que en els altres modes de joc (conquerir el castell enemic), sols que aquí aquest castell no pertanyerà a cap jugador (ni humà ni virtual) sino que serà estàtic i predefinit).

- Dos jugadors: En aquest mode de joc dos jugadors humans s'enfronten en un mapa. Els dos jugadors controlen els seus avatars per torns, i l'objectiu del mapa és conquerir el castell del jugador contrari.
- Contra la màquina: En aquest mode de joc un jugador humà juga contra un oponent controlat per la màquina i que pot actuar igual que aquest. O sigui que la màquina disposa també d'un avatar, d'un castell, i té la possiblitat de prendre mines de recursos, atacar l'avatar del jugador, o atacar el castell per intentar conquerir-lo. En aquest mode l'oponent ha d'estar dotat d'un cert grau d'intel·ligència artificial (encara que només sigui uns moviments aleatoris i uns patrons de conducta referents a construcció i reclutament de criatures, presa de mines, i atac al jugador).

Per a la realització d'aquest projecte s'utilitzarà la plataforma Visual C# junt amb les llibreries DirectX. Malgrat inicialment la idea era realitzar-lo sota C++, la major orientació a objectes i a sistema operatiu Windows del C# han fet reconsiderar la tria i prendre el C# com a llenguatge per desenvolupar-lo.

3. Eines utilitzades

Per al desenvolupament d'aquest projecte s'han fet servir bàsicament tres eines: el llenguatge de programació C# en l'entorn Visual de Microsoft, el Software Development Kit de DirectX (versió de Març del 2008), i programes per visualitzar i editar imatges.

Inicialment aquest projecte es volia desenvolupar en el llenguatge C++, però en començar a cercar informació en les etapes més primàries d'aquest un servidor va percatar-se de l'existència d'un llenguatge anomenat C# (del qual en coneixia el nom, però res més) que semblava basar-se en una estructura i un llenguatge molt similar al ja conegut C++, però que presentava una major orientació a objectes i, sobretot (i en gran part degut a la interfície Visual) una major orientació a aplicacions per Windows. Donat que el projecte a desarrollar havia de funcionar en Windows i a més pressumiblement havia de tenir un alt càrrec de orientació a objectes es va considerar la possibilitat de recomençar el projecte en aquest nou llenguatge. La gran quantitat d'informació trobada a través de la web que relacionaven el SDK de DirectX i aquest nou llenguatge van declinar finalment la balança.

Ja des d'un principi la idea era utilitzar el DirectX com a llibreria gràfica i sonora per a desenvolupar el projecte, i és que partint de la base que havia de ser un joc en 2D el DirectX era un bon punt de partida per a aquest (i també pel fet de saber a caire d'experiència personal que la gran majoria del jocs sota Windows funcionaven amb aquesta llibreria). En aquesta decisió no hi va haver cap replantejament, doncs seguint la tria inicial es va procedir a obtenir la última versió disponible del conjunt de llibreries (que es pot descarregar gratuïtament a través d'Internet) i a començar a cercar informació sobre el seu funcionament i com aquesta es podia integrar en el desenvolupament d'una aplicació (cosa que abans de començar aquest projecte un servidor no tenia ni idea de que s'hagués de fer).

Per últim, els programes per visualitzar i editar imatges van fer acte de presència més tard. Un cop entès el sistema de funcionament gràfic dels jocs en 2D (basat en *sprites* que es succeïen un rere l'altre, com el fotogrames d'una pel·lícula) va sorgir la necessitat d'obtenir un conjunt de gràfics que fós visualment agradable, que presentés una mínima qualitat, i que els diferents components

fóssin semblants entre ells. Tots els gràfics utilitzats per aquesta aplicació s'han tret directament d'Internet, i la majoria d'ells formen part d'altres jocs en 2D que funcionen amb el mateix sistema gràfic. Llavors això obligava a trobar gràfics que tinguéssin una coherència entre ells, doncs era impensable intentar moure un personatge d'un joc del 1991 sobre un fons d'un joc del 2001 (a banda que l'aspecte dels jocs del 1991 deixava molt que desitjava pel que ara coneixem). O sigui que aquí va començar una llarga cerca de material gràfic que seguís uns patrons de tamany desitjats, que fós coherent entre ell, i que no ofengués el sentit de la vista. A caire anecdòtic, abans de trobar el material que finalment s'ha fet servir (i després de moltes cerques infructíferes) un servidor es va dedicar a instal·lar un joc en 2D i realitzar captures de cada *sprite* amb el botó *Impr Pant*, per després sortir del joc, enganxar la imatge en un programa de dibuix, retallar l'*sprite* desitjat (seguint uns patrons de mides de pìxels determinats), i començar píxel a píxel a pintar de blanc el fons de l'*sprite* per tal de poder-lo interpretar llavors com a transparència en dibuixar-lo al programa. No obstant, després de passar hores obtenint els *sprites* referents a un perfil d'un personatge i un parell d'edificis (i pensant que, com a mínim, hi havia d'haver 2 personatges amb 4 perfils diferents, i com a mínim 6 edificis diferents, a part dels elements del terreny com arbres o muntanyes) un servidor va entendre que aquella no era la via cap a l'èxit, així que es va reiniciar la cerca de través de la web de material que pogués satisfer les condicions necessàries. Afortunadament al cap de poc van aparéixer a la pantalla un seguit de *tilesets* (nom que designa les imatges formades per un seguit de rajoles que s'utilitzen per dibuixar fons en els jocs en 2D) d'aspecte més que correcte, i perfectament distribuïts dins la imatge (cada rajola tenia un tamany de 32 per 32 píxels, sense excepció). L'atzar va fer que poc després d'això aparegués a la pantalla un seguit de *charsets* (que és un arxiu d'imatge que conté un seguit de personatges amb tots els seus *sprites*) d'aspecte idèntic als *tilsets* (pel que fa als colors, el tamany, l'ambientació i la qualitat), i poc després un seguit d'*sprites* estàtics que representaven criatures i, a part de ser perfectament coherents amb els *tilesets* i *charsets* trobats, anaven perfectes per al sistema de batalles per torns que el joc havia de tenir.

Així que un cop decidides les eines que s'havien d'utilitzar en el projecte, havia ja estava apunt per començar-se a dissenyar i implementar.

4. Metodologia

Malgrat la metodologia seguida en tots els passos de la creació d'aquest projecte s'explica detalladament a l'apartat d'implementació, al seguir tots uns mateixos principis aquests s'exposen aquí de forma genèrica.

El mètode que s'ha fet servir en cada un dels processos del joc i en el propi joc en sí consisteix en implementar primer un cas molt concret i limitat d'allò que es volia aconseguir, per després, un cop aquest funcionés, començar a ampliar-ne l'abast i les possibilitats. Per exemple, per aconseguir que per pantalla es mostrés l'*sprite* pertinent al personatge al principi es va restringir específicament al codi que mostrés aquell *sprite* indicant-li explícitament l'arxiu on es trobava i les posicions que en cada pas havia de mostrar. Un cop això va estar implementat i va funcionar, llavors es va procedir a "obrir" les fronteres d'aquest procés, intentant primer doncs que la ruta de l'arxiu que contenia l'*sprite* l'obtingués des de l'objecte personatge, per llavors intentar que els diferents *sprites* que havia de mostrar en caminar els obtingués de forma relativa a l'inicial (i no de forma absoluta per aquell cas concret), i finalment modificant els paràmetres perquè funcionés perfectament amb qualsevol *sprite* de personatge que li fós assignat.

D'aquesta mateixa forma es va partir d'un projecte lineal a un projecte basat en objectes, doncs el que en un principi s'havia implementat de forma absoluta per verificar-ne el seu correcte funcionament, llavors es va transformar per tal que funcionés a través de mètodes i atributs dels objectes que hi tenien relació, o bé que prengués dades des d'un fitxer (per tal de poder-les parametritzar, o simplement perquè el propi disseny del joc requeria que aquelles dades no fóssin estàtiques). Per exemple, el primer combat que es va implementar va ser contra un conjunt de criatures predefinides i estàtiques, i un cop aquest va funcionar les dades que es mostraven i que s'utilitzaven d'aquestes criatures es van començar a prendre des dels objectes referents a cada una d'aquestes criatures, per finalment parametritzar tot el conjunt de criatures i fer que aquestes es prenguéssin des de dos fitxer de text: un que determinava quines criatures hi havia d'haver en el combat, i l'altre que contenia tota la informació referent a totes les criatures existents en el joc.

Així doncs, malgrat evidentment cada procés té les seves complicacions particulars, el mètode que s'ha fet servir per cada un d'ells ha set el mateix: partir d'un cas molt particular per llavors transportar-lo a un procés general i vàlid per tots els casos possibles.

5. Funcionament del joc

Ja s'ha mencionat anteriorment que aquest joc serà un joc d'estratègia per torns en el qual el/s jugador/s controlarà un avatar que mourà per un mapa i disposarà d'un castell on podrà edificar construccions i reclutar criatures. Anem ara a ampliar aquesta descripció explicant detalladament com es desenvolupa el joc a nivell d'usuari (sense entrar en detalls referents a la implementació, sino sols el funcionament del joc). Per fer-ho, es dividirà aquest apartat en diversos sub-apartats per tal d'estructurar una mica més aquesta explicació.

Joc per torns: Un joc per torns és aquell en què les accions es succeeixen una rere l'altra enlloc de passar simultàneament. Així, mentre que en un joc no estructurat per torns el jugador i el rival realitzaran les seves accions alhora, en un joc per torns primer un jugador realitza totes les seves accions durant aquell torn, per després deixar pas a un altre jugador (o al PC, en cas de jugar contra la màquina) perquè faci el mateix. Així doncs aquesta estructuració farà que en aquest joc el jugador que primer prengui el torn pugui moure l'avatar, entrar al seu castell per construïr o reclutar criatures (més endavant es detalla concretament el què es podrà fer dins el castell), interactuar amb el mapa per on es mogui, o iniciar una batalla, per finalment un cop finalitzat el seu torn cedeixi el torn al següent jugador o a la màquina. S'ha de tenir en compte que les batalles o combats funcionen igualment mitjançant un sistema de torns (detallat també més endavant). Aquests torns, doncs, s'han de limitar d'alguna manera. Cada avatar tindrà a l'inici del seu torn una sèrie de *punts de moviment o acció*. Aquests limitaran el nombre d'accions que l'avatar pot fer per un torn, i és que cada moviment o acció realitzada (entenent per acció entrar al castell, iniciar una batalla o prendre una mina) consumirà un punt de moviment, de forma que quan l'avatar es quedi sense no podrà fer res més fins que torni a ser el seu torn, moment en què s'he l'hi reestabliran els punts de moviment a la quantitat inicial.

Mapa: L'acció del joc es desarrollarà sempre sobre un mapa. Aquest serà un espai limitat de terreny on es trobaran els castells i avatars dels diferents jugadors, així com tots els elements amb els quals puguin interactuar els avatars. Aquests mapes tindran zones per les quals es avatars es podran moure, zones a

través de les quals no es podrà passar, i elements amb els quals es podrà interactuar (com els castells, els altres avatars, les mines o enemics).

Jugador: Cada persona que participi al joc (o màquina en el seu defecte) s'anomenarà jugador, i tindrà un nom, un color determinat, un castell i un avatar. El nom del jugador no tindrà cap efecte dins el joc, mentre que el color servirà per "marcar" les mines que el jugador prengui amb l'avatar (el sistema de mines està explicat més endavant). Cada jugador tindrà un únic castell a on podrà realitzar totes les accions pertinents a aquest (també explicades més endavant), i tindrà un sol avatar. Amb aquest es desplaçarà pel mapa i interactuarà amb els diferents elements.

Recursos: Cada jugador tindrà una sèrie de recursos amb els quals podrà edificar noves construccions en el seu castell o reclutar criatures, entre d'altres. Aquests recursos s'estructuraran en dos petits blocs: hi haurà un recurs primari que actuarà a mode de diners o or, i 3 recursos secundaris que actuaran a mode de matèries primeres per la construcció. Cada torn el jugador en qüestió guanyarà una determinada quantitat de diners, mentre que per guanyar recursos secundaris el jugador haurà de posseïr una o més mines d'aquest recurs (el funcionament de les mines s'explica més endavant). Així doncs si el jugador no disposa d'una quantitat determinada de recursos no podrà construïr un determinat edifici, o no podrà reclutar un nombre determinat de tropes i haurà d'esperar a tenir aquests recursos.

Castell: Cada jugador tindrà un castell i aquest serà la base de la seva supervivència. Perdre el castell significarà perdre la partida. En aquest castell el jugador podrà construïr uns determinats edificis (en cas que disposi dels suficients recursos) per després poder reclutar unes determinades criatures. Cada nou edifici proporcionarà un nou tipus de criatura per reclutable, essent cada una més poderosa que l'anterior. Per reclutar criatures el jugador haurà de disposar de la quantitat de recursos suficients, i el nombre que se'n podrà reclutar estarà limitat per un nombre de torns en concret. O sigui que les criatures disponibles no seran il·limitades en nombre, sino que cada *n* torns hi haurà *k* criatures disponibles, i es podran reclutar totes, algunes, o cap, de forma que al cap de *n* torns més el nombre de criatures que estiguin disponibles augmentarà en *k*. Aquests criatures podran passar a formar part de les criatures que acompanyin l'avatar o quedar-se al castell a mode de defensa. En cas que un jugador contrari intenti apodarar-se

del castell, si en aquest hi ha defenses s'iniciarà una batalla. Si el jugador contrari guanya la batalla, o no hi ha defenses al castell, el jugador contrari s'apoderarà del castell, de forma que finalitzarà la partida.

Avatar: L'avatar serà un personatge que cada jugador crearà abans de començar la partida (escollint el seu nom i el seu aspecte). Aquest tindrà a la seva disposició un seguit de criatures amb les quals podrà combatre (més endavant s'explica com funcionen les criatures). L'avatar directament no combatrà mai, sino que serà sols la representació de l'entitat que es mou pel mapa (a efectes interns del joc l'avatar es pot considerar com el comandant de tot un seguit de tropes que les dirigeix en la batalla però que no hi actúa). Per tant aquest haurà d'anar acompanyat sempre de, com a mínim, una criatura. En cas de quedar-se sense es considerarà que l'avatar s'ha retirat del camp de batalla, i restarà al castell disponible per tornar a sortir al mapa (amb un cost determinat de recursos del jugador).

Criatures: Hi haurà un determinat nombre de criatures que un jugador podrà aconseguir a través d'edificacions en el seu castell, i aquestes podràn acompanyar a l'avatar o quedar-se al castell. Les batalles les durant a terme les criatures, de forma que si s'enfronten dos avatars la batalla serà entre les criatures que aquests tinguin amb ells. Les criatures s'agruparan per tipus de forma que un avatar que posseeixi *n* criatures d'un tipus determinat les controlarà en forma de grup (no es podrà controlar individualment diferents criatures d'un mateix tipus) i les accions que aquestes realitzin en un combat o els atacs que rebin es determinaran en forma de grup (o sigui, que totes realitzaran la mateixa acció determinant la seva eficàcia segons el nombre que formin el grup i totes rebran un atac determinat de forma que pereixeran algunes d'elles i el nombre que forma el grup es reduirà). Hi haurà criatures que seran més poderoses que altres, i això es determinarà principalment pels seus atributs i per les accions que podran dur a terme en un combat (no obstant el nombre de criatures que formin el conjunt tindrà efectes notables en les seves accions, de forma que encara que una criatura d'un tipus B sigui més poderosa que una d'un tipus A a efectes pràctics 100 criatures del tipus A seran més poderoses en conjunt que 10 criatures del tipus B).

Batalles: En el joc es presentaran tres situacions en les que s'iniciarà una batalla: la primera d'elles serà quan un avatar s'enfronti a un conjunt de criatures que es trobaran disperses pel mapa (que pot ser que, per exemple, guardin

l’entrada a una mina). En aquest cas s’enfrontaran les criatures que acompanyin l’avatar amb les criatures que es trobin en aquell punt del mapa. El segon cas és quan dos avatars es trobin al mapa. En aquest cas s’enfrontaran les criatures que cada un posseeixi. I el tercer i últim cas serà quan un jugador intenti apoderar-se d’un castell aliè i en aquest hi hagin defenses, moment en que s’enfrontaran les criatures de l’avatar amb les defenses del castell. Les batalles, així com la resta del joc, es desarrollaran per torns. Començarà atacant la persona que hagi iniciat la batalla, i des de la primera fins l’últim criatura al seu poder escollirà l’acció que durà a terme i sobre quina criatura la durà a terme. Un cop totes les criatures hagin atacat, llavors el torn passarà a l’altre jugador o a la màquina en funció del tipus de batalla que sigui. Aquest nou torn es desarrollarà d’igual manera, permetent a les criatures dur a terme un seguit d’accions. Una batalla acabarà quan un dels dos participants perdi totes les criatures. Si aquestes són criatures pertanyen a un avatar, aquest es retirarà del mapa i passarà a trobar-se al castell, de forma que si es vol tornar a portar al mapa s’haurà de pagar una quantitat determinada de recursos.

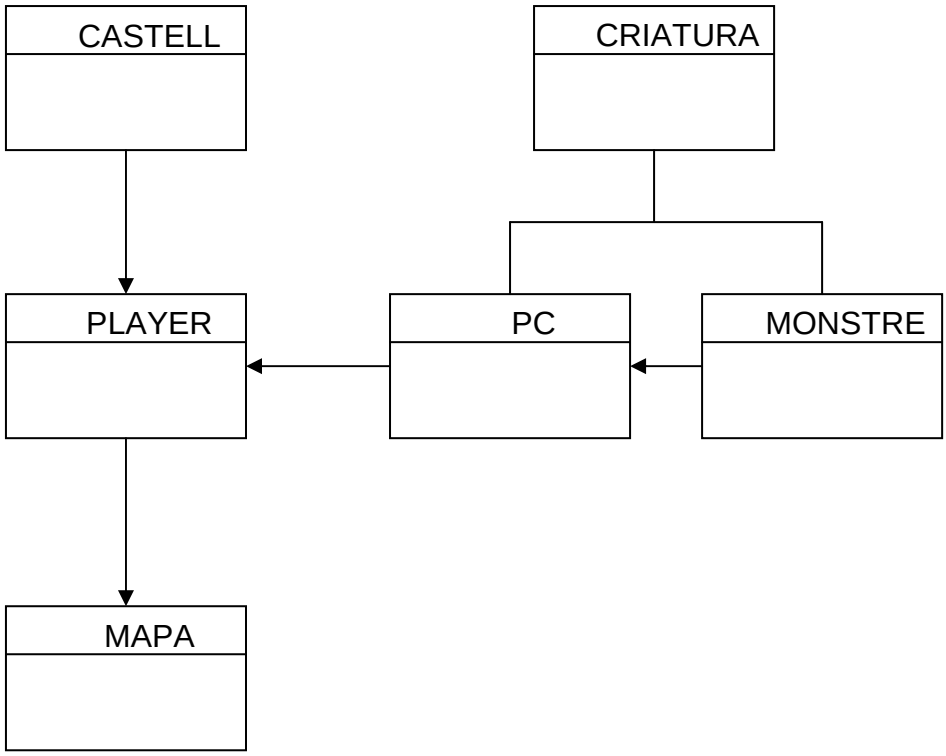
Mines: Tal i com ja s’ha esmentat en anteriors punts repartides pel mapa hi haurà un seguit de mines. Aquestes anomenades mines seran punts concrets que es diferenciaran al mapa i on un avatar podrà dirigir-se i apropiar-se’n. Quan això passi, el jugador en qüestió passarà a tenir una mina més del tipus determinat, cosa que significarà que cada nou torn el jugador guanyarà la quantitat de recursos determinats que una mina ofereixi. Aquestes quantitats seran acumulables, de forma que si un jugador posseeix dues mines d’un recurs determinat, guanyarà el doble del que proporciona una mina, i així amb totes les que tingui. No obstant, el fet que un avatar s’apoderi d’una mina no fa que aquesta sigui sempre propietat del jugador, i és que l’avatar del jugador contrari pot d’igual manera apropiar-se de qualsevol mina, sigui o no propietat d’un altre jugador. Òbviament, quan un jugador s’apoderi d’una mina que pertanyia a un altre, serà aquest primer qui en rebrà els beneficis a partir d’aquest moment i fins que la mina deixi de ser seva.

6. Implementació

En aquest apartat de la memòria és on s'explica tot el procés de creació del joc. Es divideix en unes parts lògiques, agrupant cronològicament la feina que s'ha dut a terme. Un cop explicat el funcionament del joc, es procedeix a dissenyar un esboç de com s'ha d'estructurar aquest segons els objectes que formen una partida, i què ha de tenir *a priori* cadascún d'aquests.

Pre-implementació: Estructura dels objectes que formen una partida

Partint del sistema que de joc (descrit en l'apartat *Funcionament del joc*) hi ha una sèrie de components clau que des del primer moment es va tenir clar que havien d'existir. Així doncs, aquest primer disseny genèric dels objectes que prenen part en una partida resulta en el següent:



Cada una d'aquestes classes està formada per una sèrie d'atributs i una sèrie de mètodes, que en un primer moment es va considerar que serien els següents:

CLASSE: Criatura
ATRIBUTS: Nom, imatge
MÈTODES: Mostrar (un per cada atribut)
DESCRIPCIÓ: Generalització destinada a posar en comú els atributs que comparteixin els objectes Criatura i PC.

CLASSE: PC (Player Character)
ATRIBUTS: Face, color, num_criatures, monstres[], moviment, posicioX, posicioY
MÈTODES: Mostrar (un per cada atribut), afegir_criatura, moure, passar_torn
DESCRIPCIÓ: L'objecte PC és a efectes pràctics l'avatar a qui controla el jugador i tindrà per tant un nom, un rostre, i una posició en el mapa, així com un seguit de criatures sota el seu control (o sigui, un *array* d'objectes Monstre).

CLASSE: Monstre
ATRIBUTS: Atac, defensa, HP, HP_base, MP, MP_base, quantitat
MÈTODES: Mostrar (un per cada atribut), atacar, rebre_atac, màgia (¿?), mort
DESCRIPCIÓ: Aquest objecte està destinat a representar cada un dels monstres que pot controlar el jugador o contra els que s'enfronta. A part dels atributs que ha de tenir cada una d'ells i que fan que un sigui més poderos que un altre, hi ha una sèrie d'atributs que, degut al funcionament del joc, són necessaris, com per exemple l'HP_base, que fa que en rebre un atac, si aquest supera l'HP_base, es resti directament una criatura del grup (recordant que les criatures s'agruparan per conjunts de criatures del mateix tipus).

CLASSE: Castell
ATRIBUTS: Nom, classe, imatge, color, num_defenses, defenses[]
MÈTODES: Mostrar (un per cada mètode), afegir_criatura, construïr, reclutar_criatura
DESCRIPCIÓ: Cada jugador té un castell i, com ja s'ha explicat anteriorment, la pèrdua d'aquest suposa el final de la partida. En aquest castell el jugador pot reclutar criatures per sumar-les a l'exèrcit del l'avatar o per deixar-les defensant el castell, o construïr edificis que li permetin reclutar criatures més poderoses. Les defenses del castell són de vital importància, doncs són la única barrera entre un avatar enemic i el castell del jugador.

CLASSE: Player
ATRIBUTS: Nom, color, avatar, castell, mines, recursos
MÈTODES: modificar_num_mines, modificar_recursos
DESCRIPCIÓ: Aquest objecte representa cada un dels jugadors que participen en una partida. Cada jugador té a la seva disposició un avatar i un castell, i és en aquest objecte on aquests es guardaran. Així mateix, en aquest objecte es porta el control del nombre de mines de cada recurs de les que disposa el jugador, així com de la quantitat de recursos dels que disposa, i és aquí on es reflecteix el guany o la pèrdua de mines o recursos.

CLASSE: Mapa
ATRIBUTS: Tilesets, mida_mapa, descripció, capes_mapa
MÈTODES: Carregar_mapa
DESCRIPCIÓ: El mapa és per expressar-ho d'alguna forma el taulell de joc, i per tant és aquí on tots els altres objectes interactuen. Així doncs en l'objecte mapa s'emmagatzemen totes les dades referents al terreny, a l'aspecte del terreny (a través dels tilesets), als edificis o events que hi ha en ell, així com un sistema ideat inicialment que fa que el mapa estigui basat en una sèrie de capes sobreposades (aquest sistema s'explica més endavant).

Així doncs aquest és el diagrama inicial on s' identifiquen i es defineixen els objectes que ja des d'un principi es presenten indispensables per tal de donar sentit al joc. Un cop fet això, el següent pas és ja la primera presa de contacte amb el llenguatge de programació i el conjunt de llibreries del DirectX.

Implementació – 1: Presa de contacte entre el C# i el DirectX

En un anterior apartat d'aquesta memòria ja s'exposa el perquè de la decisió de programar el projecte en C# (i no en C++ com al principi s'havia decidit) i de fer servir les llibreries del DirectX, així que en aquest punt s'entra directe en com es fa per integrar les llibreries en el llenguatge.

El primer que s'ha de fer és buscar un SDK del DirectX (o sigui un paquet que contingui totes les llibreries per poder-les utilitzar en la implementació de programes), que es troba sense gaires problemes a la pàgina de Microsoft

disponible per descarregar gratuïtament. Per ser la última versió distribuïda fins al moment s'ha escollit la versió de Març del 2008. Un cop descarregat l'SDK i obert l'entorn de programació Visual C# comencen les preguntes: i ara? Després de buscar a través d'Internet com es pot implementar un programa utilitzant DirectX apareix una pàgina on s'explica perfectament els passos a seguir. Cada projecte de C# té una sèrie de referències que s'hi poden vincular per tal de poder-les utilitzar posteriorment, i un cop descarregat l'SDK i buscades les llibreries es volen incloure al projecte apreixen tot el seguit de paquets que inclou el DirectX (entre d'altres el Direct3D, el DirectPlay, el DirectSound, etc). Inicialment s'inclou només la llibreria Direct3D (ja que, malgrat el joc sigui en 2D, el sistema utilitzat per mostrar els gràfics per pantalla és el Direct3D ja que presenta un sistema de funcionament curiosament més senzill que la llibreria per mostrar gràfics en 2D) i seguidament es va procedeix a declarar i definir les variables i els dispositius que s'han d'utilitzar.

El funcionament es basa en definir un dispositiu anomenat *device* (que representa la superfície sobre la qual es mostraran els gràfics, ja n'existeixi una o més) al qual se li assigna un tamany determinat i una superfície sobre la qual mostrar-se (per exemple, sobre una finestra de Windows normal i corrent). Malgrat és possible definir un *device* de 3 maneres diferents existeix una forma per definir-lo que simplifica en gran mesura el seu ús encarat a gràfics en 2D (ja que no necessita tants paràmetres i especificacions com un *device* encarat als gràfics en 3D), i pràcticament només hi ha dos paràmetres rellevants: el primer que indica el dispositiu físic que s'utilitzarà per mostrar el *device* (per defecte la pantalla), i el segon que indica el tipus de dispositiu que es vol crear per mostrar els gràfics (però que utilitzant el mètode DirectX.Direct3D.DeviceType es crida un rastrejador que automàticament busca el dispositiu).

Un cop definit el dispositiu el primer pas que s'ha de dur a terme és definir tot un seguit de variables i mètodes que converteixen la finestra en la interfície on es mostrarà el resultat del mètode per pintar la pantalla. La variable fonamental és crear un *device* per poder-se referir lògicament, i que abans de començar a mostrar gràfics haurà d'estar definit correctament.

Com a mètode de la classe finestra on es treballi es defineix una acció anomenada InicialitzarGràfics que configura tots els paràmetres i opcions referents al *device* que més endavant s'utilitzarrà per mostrar els gràfics. No s'entrarà en

detall de la definció d'aquests (es basen en definir la superfície del *device*, el seu tamany, si acceptarà transparències, etc.).

Un cop definit aquest mètode cal definir el mètode que s'encarregarà de pintar un cop rere un altre la pantalla (per tal de reflectir tots els canvis que hi passin). Aquest mètode (anomenat OnPaint) s'encarrega de borrar tot el que hi ha al *device* per després tornar-ho a dibuixar tot de nou.

Finalment i per tal que es puguin utilitzar els dos mètodes nombrats s'han de cridar des del mètode *main*, cridant primer al mètode InicialitzarGràfics, per tal que després el mètode OnPaint es vagi repetint contínuament per redibuixar qualsevol cosa que canvii a la pantalla.

Un cop fet tot això el programa ja es pot executar i mostra una finestra de Windows completament pintada de negre. Per algun lloc es comença, i ara doncs és el moment de convertir aquest negre en quelcom més.

Implementació – 2: Explorant el Direct3D

2.1: Explorant el Direct3D: dibuixar textures

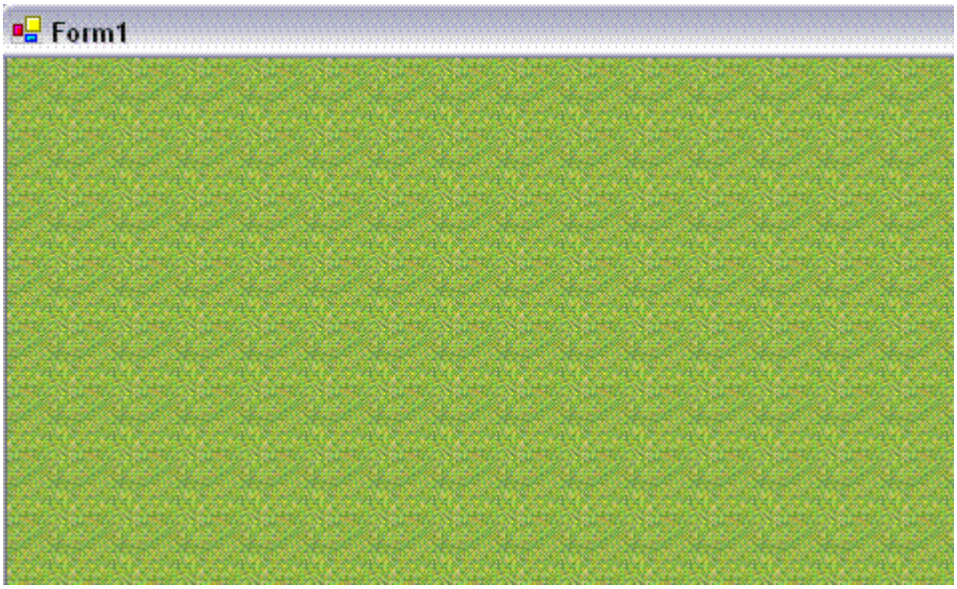
Qualsevol joc en 2D (exceptuant el mític joc de recreativa de matar naus espacials, o el de la pilota i les dues "raquetes") utilitza textures. S'anomena textura a una imatge que sovint representa una superfície d'un cert material i que, posada conjuntament amb ella mateixa, sembla que sigui una única superfície enorme. O sigui que és com una porció de material que en cada un dels seus límits presenta un aspecte que concorda amb el límit oposat, de tal forma que no presenta un tall accentuat si es posa la mateixa textura un cop rere un altre formant una fila, o formant tota una superfície. Aquestes textures s'utilitzen per representar terreny, per exemple, doncs en el cas del joc a realitzar hi ha d'haver una superfície per on es mooguin els avatars que representarà per exemple prats, boscos o muntanyes. Així doncs, calen textures per representar herba, roca o arbres. Partint d'una textura d'herba extremadament simple (formada per un *tile* de 32 x 32 píxels, recordant que un *tile* és com s'anomena a aquesta imatge que es repetirà un cop rere un altre per simular una extensió de terreny major al seu tamany) es pot aconseguir que tota la pantalla presenti l'aspecte d'una enorme superfície d'herba. Així doncs, el següent pas és definir una textura i dibuixar-la.

L'objecte *Texture* forma part d'una subllibreria del Direct3D anomenada Direct3DX. Així doncs és necessari agregar una referència a aquesta llibreria. Un cop fet això s'ha de declarar una variable *Texture* (declarada al mateix lloc on es declara qualsevol variable utilitzada en el programa), per després definir-la en el mètode InicialitzarGràfics. D'entre paràmetres que necessita el mètode per definir textures se'n destaca un d'especialment important que és el fixter d'imatge des d'on es prendrà la textura. Aquest fitxer pot ser per exemple una imatge de 32 x 32 píxels que representi aquesta superfície d'herba. Aquesta imatge serà doncs la que es pintarà quan es faci referència a pintar la textura en qüestió. A part d'això com a paràmetres del mètode també en destaca un que serveix per indicar si hi haurà color per a representar transparència i, en cas afirmatiu, quin serà aquest color. Això és així perquè pot interessar que una textura concreta presenti una transparència per mostrar el que té a sota (aquí és on prendrà sentit el sistema de mapes per capes que s'explicarà més endavant), i definint un color determinat el programa farà que aquell color en concret no es pinti, sinó que quedi a la vista el que tindrà sota.

Aquesta textura, quan s'hagi de pintar a la pantalla, ho haurà de fer a través del que s'anomena un *sprite*. Es podria dir que els sprites són com capes que es pinten amb les textures que s'hi associen. Per tant, així com s'ha declarat i definit una textura, s'ha de fer també amb l'*sprite*.

Un cop declarada i definida aquesta nova variable *Texture*, junt amb el seu corresponent *sprite*, arriba el moment de pintar-la. Això s'ha de fer al mètode OnPaint de tal forma que qualsevol canvi que es produeixi en qualsevol objecte que es mostri a la pantalla quedi reflectit a l'acte. El sistema de pintar a través de *tiles* fa que, per pintar la pantalla sencera, sigui molt útil implementar, per exemple, un bucle que "escombri" tota la pantalla dividint-la en petits trossos del tamany d'aquest *tile*. Un cop creat aquest bucle, doncs, només falta cridar el mètode *Sprite.Draw* a cada volta del bucle, passant-li com a paràmetres rellevants la textura que s'utilitza per pintar, el tamany del *tile*, i la posició des on ha d'anar a buscar l'origen del *tile* en la imatge (en el cas d'una imatge que representi un sol *tile* aquest origen serà sempre el punt 0,0, però això pren sentit quan s'utilitza un anomenat *tileset*, que és una imatge que conté varis *tiles* dins seu i que, per tant, per cada un s'ha d'indicar on comença), així com evidentment la posició a la pantalla on s'ha de pintar i el color que s'ha d'interpretar com a transparència.

Un cop fet això el resultat que es mostra en iniciar el programa és una finestra de Windows completament pintada amb el *tile* utilitzat, de forma que presenta un aspecte com el següent:

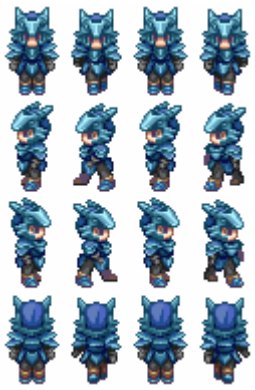


2.2: Explorant el Direct3D; dibuixar un personatge

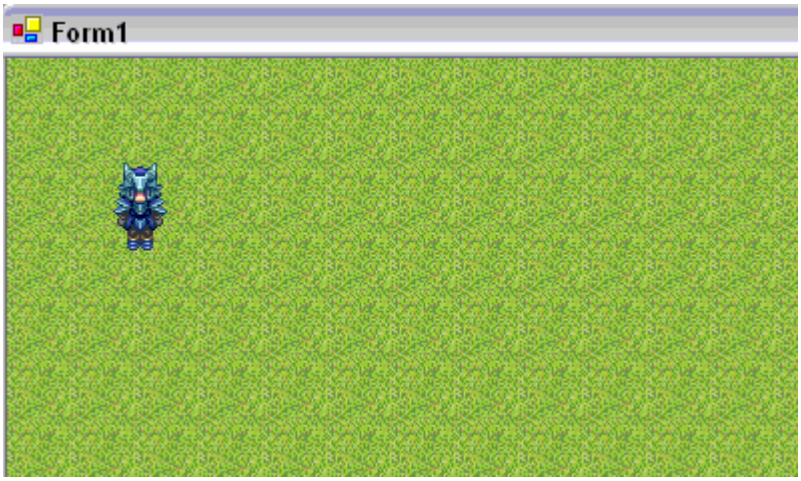
Un cop vist el funcionament de les classes *Texture* i *Sprite* és hora d'anar un pas més enllà: dibuixar un personatge a la pantalla. Sabent com es pot definir una textura, i com es pot dibuixar a la pantalla un cop definida, no és molt difícil pensar en com es pot definir la textura d'un personatge de la mateixa manera que s'ha fet amb la textura d'herba i llavors pintar-la a la pantalla tenint en compte, això sí, que a diferència de la textura anterior, el personatge no s'haurà de pintar cobrint tota la pantalla, sinó només en un punt concret.

La textura del personatge que s'utilitzarà per al següent pas presenta un seguit de rectangles blancs amb diferents posicions del personatge. El perquè d'aquest seguit de rectangles en una mateixa imatge és molt senzill, i és que igual que amb el que abans hem nombrat com a *tileset*, existeixen unes imatges anomenades *charset* i que estan formades per un o més personatges en totes les posicions amb les que pot aparèixer. No obstant això prendrà més rellevància quan sigui l'hora de moure al personatge. Ara per ara, el que sí és rellevant és el fons blanc que rodeja al personatge, i és aquí on per primer cop prendrà sentit el paràmetre per indicar el color de transparència de la textura, doncs si no indiquem

en la definció d'aquesta que el blanc s'ha d'interpretar com a transparent el personatge ens apareixerà rodejat d'un rectangle blanc. Així doncs, d'igual manera que s'ha fet amb la textura anterior s'ha de declarar i definir una nova textura que prengui la imatge del fitxer que contingui el personatge i que interpreti el color blanc com a transparència. Un cop definida aquesta nova textura és hora de pintar-la (utilitzant el mateix mètode OnPaint) i, tenint en compte que l'ordre per pintar les textures importa (ja que se'n pintaran unes sobre d'altres), s'haurà d'escriure el mètode per pintar el personatge després del mètode utilitzat per pintar la textura del terreny i, evidentment, pintar el personatge una sola vegada enlloc de cobrir tota la pantalla amb la seva imatge. Donat que la imatge del personatge no té una mida de 32 x 32 píxels (sino que és de 48 x 32) s'ha d'indicar a l'hora de pintar la textura, de forma que enlloc de pintar un rectangle de 32 x 32 en pinti un de 48 x 32.



Un cop implementat aquest nou canvi, en executar el programa es veu el fons preexistent pintat amb la textura d'herba i el personatge tal com es mostra seguidament:



Ara la finestra ja mostra un terreny i un personatge sobre ell. El proper pas és convertir el personatge en un objecte mòbil.

Implementació – 3: Moviment del personatge

3.1: Moviment del personatge; direcció

Fer que el personatge sigui un objecte mòbil es pot dividir inicialment en dues accions diferents: fer que el personatge es desplaci a través de la pantalla, o fer que el personatge canviï el seu aspecte (per exemple, girant sobre ell mateix). Inicialment és millor començar amb la segona acció, ja que *a priori* la major part dels coneixements que es necessiten ja s'han vist en els altres apartats.

Per tal de fer que el personatge pugui moure's a petició de l'usuari, s'ha de trobar alguna forma per interactuar amb els dispositius d'entrada (com per exemple el teclat). Aquestes accions s'inclouen en la llibreria DirectInput del DirectX, i és mitjançant aquesta que es pot definir un dispositiu *device* que, per tant, s'ha de diferenciar del dispositiu *device* provinent del Direct3D indicant tota la seva procedència. A aquest nou dispositiu se l'hi ha d'assignar un dispositiu d'entrada des d'on ha de capturar els esdeveniments. Aquest dispositiu d'entrada és el teclat i, un cop definit, es disposa d'un mètode per obtenir les tecles que es premen. A continuació s'ha d'implementar un mètode que diferencïi entre les diferents tecles que l'usuari pugui teclejar i dóni per cadascuna d'elles la resposta esperada. Així doncs, la idea és que en prémer cada una de les tecles de direcció el personatge miri cap a aquella direcció. Això es pot implementar a través d'una estructura on el programa entri cada cop que es premi una tecla, i on dins d'aquesta es pugui donar una acció a unes tecles en concret (comprobant a través de condicionals si la tecla que s'ha premut pertany a alguna de les tecles de direcció). Per tal que el personatge respongui a aquestes tecles canviant la seva direcció s'ha de canviar la posició on el mètode *Sprite.Draw* va a buscar el personatge dins la imatge. Així doncs, aquesta posició ha de trobar-se dins una variable que es modifiqui en prémer alguna de les tecles de direcció i que li doni el valor de la posició on comença el *tile* del personatge mirant cap a aquella direcció en concret.

Així doncs, un cop implementat això el programa mostra de nou la finestra de Windows completament pintada amb el *tile* que representa herba i amb el personatge, però amb la novetat que en prémer alguna de les tecles de direcció el personatge mira cap a la direcció indicada.

3.2: Moviment del personatge; desplaçament per la pantalla

Fins aquí el personatge ja reacciona a unes entrades a través del teclat. No obstant, un personatge que simplement canvia el seu punt de direcció no proporciona una base molt suculenta per a un joc. Així doncs, és necessari que el personatge es mogui, que es desplaci a través de la pantalla. Un cop arribats a aquest punt, coneixent com es dibuixa el personatge i veient que és possible capturar tecles i fer reaccionar el personatge a elles, és pressumible com es podrà implementar que el personatge es mogui. El mètode per dibuixar el personatge utilitza una dada que indica la posició de la imatge a mostrar dins el fitxer, i aquesta és modificable a través del teclat. Sabent doncs que aquest mateix mètode també utilitza un paràmetre per indicar on ha de dibuixar el personatge dins la pantalla, també es pot suposar que és possible modificar aquesta dada a través d'unes entrades determinades del teclat. Per exemple, i ja que els *tiles* que s'utilitzen són de 32 x 32, es pot indicar la posició on es dibuixa el personatge a través d'una variable, i aquesta es pot modificar amb cada tecla que s'apreti, de forma que si el personatge es mostra inicialment en una determinada posició X, es pot fer que en presionar la tecla dreta, per exemple, aquesta posició prengui el valor de X+32, de forma que el personatge es mostri just després d'apretar la tecla a "un *tile*" cap a la dreta d'on era abans, i així mateix amb totes 4 tecles de direcció. Un cop assolits els coneixements derivats dels apartats anteriors, aquest punt no té molta dificultat. L'únic problema que hi ha és que considerant la unitat de desplaçament com a 32 píxels el personatge es mou a la velocitat d'un llamp, cosa que ni és creible ni és estètic. Modificant doncs aquesta mesura que es suma a la posició per (per exemple) 4 enlloc de 32 s'arregla aquest problema. L'únic que queda de moment referent al personatge és que aquest es mogui mentre es desplaça (ja que de moment el seu desplaçament el fa levitant).

3.3: Moviment del personatge; animació

Tenint el *charset* mostrat anteriorment es pot veure que es disposa de les imatges del personatges mirant en les 4 direccions diferents i en 4 posicions diferents, que no són més que les 4 posicions que representen al personatge caminant. Si es vol animar al personatge el que s'ha de fer és quelcom semblant al que s'ha fet per fer-li canviar el punt de vista i per fer-lo desplaçar. Sabent que un

dels paràmetres que utilitza el mètode per pintar la textura és el punt de la imatge on va a buscar al personatge (cosa que ja s'ha utilitzat per fer que aquest canvïi de direcció), es pot a continuació modificar no sols la fila on va a buscar la textura, sino també la columna (tenint en compte que a cada columna hi ha una posició diferent del personatge). Així doncs, implementant que per cada tecla que es premi amb una mateixa direcció la columna avanci una "posició" (que seran 32 píxels, l'amplada del personatge) i tenint en compte que un cop arribat a l'última s'ha de tornar al principi, el personatge no sols canviarà de direcció i es desplaçarà, sino que també es podrà veure com en moure's camina.

Amb això el personatge ja és capaç de realitzar tot el que gràficament s'espera d'aquest, amb un petit problema: donat que reacciona a cada tecla i només a cada tecla, és possible moure el personatge una unitat extremadament petita i deixar-lo a mitja passa, per seguir-lo movent a partir d'allà, fent això no sols que el personatge no respecti la quadrícula de 32 x 32 que s'havia pres com a unitat mínima, sino que sovint queda en posicions extranyes. Com es pot solucionar això? S'ha d'obligar a que, per cada tecla que es premi, el personatge es mogui una mida mínima (corresponent a un *tile*) i, per tant, s'ha de vigilar també que mentre estigui executant aquest moviment no es pugui alterar amb altres tecles de direcció. Això es pot aconseguir si per cada tecla es crea un bucle que, per exemple, faci moure al personatge 4 píxels durant 8 vegades, de tal forma que mogui un total de 32 píxels i passi la seva animació completa 2 vegades, de forma que acabi el moviment en la posició de repòs que l'ha iniciat. Tal vegada, per evitar que aquest moviment pugui ser interferit a mig fer per altres tecles, es pot declarar un booleà (anomenat "moviment", per exemple) que en moure el personatge es posi a cert, i fer que el programa només capturi tecles quan el booleà sigui fals. D'aquest forma el prémer una tecla fa que el programa entri en un bucle que es troba al mateix mètode OnPaint (d'aquesta forma s'anirà repetint mentre duri) i és dins aquest on, a cada volta, es modifiquen els atributs de posició i imatge del personatge per, al final, posar el booleà a fals i permetre obtenir una nova direcció. En un primer moment aquest bucle es pot pensar com un *while*. Problema? Donat que aquest *while* es troba dins el mateix mètode OnPaint, per cada moviment queda pintada tota l'"estela" d'imatges del personatge, una rere l'altra sobreposant-se a 4 píxels de distància. Solució? Aprofitant el bucle "moviment" creat anteriorment, i sabent que el mètode OnPaint s'executa contínuament una vegada

rere una altra, es pot crear un condicional on el programa entri en cas que “moviment” sigui igual a cert, i dins aquest condicional crear un comptador que el faci executar 8 vegades (canviant la posició i la imatge a mostrar en cada una d’elles) per al final posar el booleà a fals.

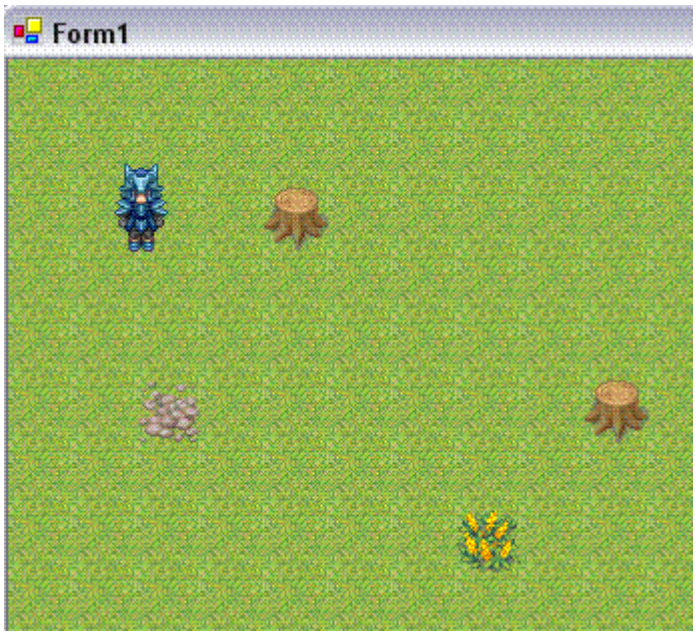
Amb això doncs el resultat és una finestra de Windows pintada completament amb una textura semblant a herba i sobre la qual es mostra un personatge que, en prémer alguna de les tecles de direcció, canvia el seu punt de vista i es mou una unitat mínima preestablerta animant el seu moviment i impedint que aquest sigui tallat a mitja execució per una nova entrada del teclat. Un cop arribat aquí ja es disposa d’una base mínima per a començar a crear un terreny de joc. El pròxim pas és donar varietat a aquest entorn per on mou el personatge.

Implementació – 4: Terreny i obstacles

Si el joc que es volgués implementar fós un joc de futbol segurament amb la textura d’herba ja quasi en tindriem prou. Però no és així, i es necessita poder crear un entorn variat i atractiu per on poder moure el personatge. Anteriorment s’ha parlat d’unes imatges de tamany superior a un *tile* anomenades *tilesets* que no són res més que un conjunt de *tiles* (igual que la imatge mostrada del personatge, on aparèixen totes les posicions possibles d’aquest, però mostrant diferents tipus de terreny). És doncs en aquests *tilesets* on s’ha d’anar a buscar varietat pel terreny. Amb el que s’ha vist fins ara, sabent que és possible modificar la posició a partir de la qual el programa pren una textura dins una imatge, es pot suposar doncs que el mateix es pot fer amb un conjunt de *tiles*, de forma que en pintar el mapa a través del bucle *for* es poden pintar textures des de diferents posicions dins un *tileset*. A través d’Internet es troben multitud d’aquestes imatges (algunes d’elles tretes d’altres jocs de PC i algunes de creació pròpia), i una d’elles és la següent:

Aquí tornen a prendre vital importància les transparències, doncs tot el color de fons que presenta la imatge ha de ser en realitat una transparència. La primera reacció instantànea és pintar conjuntament *tiles* que mostren herba i *tiles* que mostren, per exemple, la soca d'un arbre. El problema d'això és que veient el *tile* corresponent a la soca d'un arbre es dedueix que aquest ha de presentar una transparència, i si aquesta ja es pren com a tal i el *tile* es dibuixa conjuntament amb el altres *tiles* que formen la base del terreny (en aquest cas herba) el que es mostra entre la soca i l'herba no és mès que el que hi hagi sota, o sigui, negre. Aquí sorgeix la primera necessitat de pensar el mapa com una sèrie de capes, doncs no és equívoc pensar que si es pinta tot el mapa amb herba i després a sobre es pinta la soca de l'arbre el que es veu darrere és herba, tal i com toca. Així doncs aquí neix aquesta idea, que donarà molt fruit al llarg de tot el projecte. Per a dur-la inicialment a terme el que es fa primerament és deixar inalterat el bucle que pinta completament la pantalla amb el *tile* corresponent a herba. Seguidament el que s'ha de fer és pintar algunes posicions de la pantalla amb els tiles corresponents que s'hi vulguin pintar (per exemple el ja esmentat *tile* que representa una soca, un que representi flors i un que representi pedres). No obstant d'alguna manera és necessari determinar en quines posicions es pintaran aquests nous *tiles*, i és que un bucle sencer (com el de l'herba) ompliria absolutament tota la pantalla. Una primera idea és generar un nombre aleatori i llavors passar-lo per un *switch* que en determinats valors no pinti res, en d'altres una soca, en d'altres les flors, etc. Problema? Donat que el bucle OnPaint passa contínuament a cada nova volta els valors d'aquest nombre aleatori canviaran, de forma que en executar el programa presentaran un entorn amb infinites intermitències de soques, flors i pedres. És necessari, d'alguna manera, determinar de manera estàtica les posicions que correspondran a cada *tile*. Sabent

que cada *tile* que utilitzem pel terreny tindrà una mida de 32 x 32, i que segons aquesta mida la pantalla queda dividida en una cuadrícula de 32 x 24, la solució és crear una matriu de 32 x 24 de, per exemple, enters per llavors, un cop passat el bucle que pinta el terra, passi totes les posicions de la taula amb un *switch* que consideri, per exemple, no pintar res si hi ha un 0, pintar una soca quan hi hagi un 1, les flors quan hi hagi un 2, i les pedres quan hi hagi un 3. Amb aquesta matriu doncs, i ja implementat el corresponent bucle que la recórre, l'aspecte del programa és el següent:



Amb això neix la primera capa del mapa; una capa d'enters que es sobreposa a una capa base, en aquest cas tota formada pel mateix *tile*. Amb aquesta introducció al sistema de capes ja es pot intuïr tot el potencial que té, i és que a partir d'aquí es dividirà el mapa en, per exemple, capes de terreny, d'obstacles, d'edificis i, per què no, de personatges.

Ara el mapa presenta un aspecte més variat i atractiu i, com en quasi tots els canvis en el món de la informàtica, té els seus avantatges i els seus desavantatges. L'inconvenient és que, malgrat l'ésser humà intueix que el personatge no hauria de poder caminar per sobre la soca de l'arbre, l'ordinador no ho perceb així, de forma que el personatge camina alegrement per qualsevol part del mapa. Aquí és on neix la necessitat d'implementar un anomenat sistema de detecció de col·lisions, que no és més que un sistema que s'ha d'encarregar

d'avançar-se a la següent posició on s'ha de moure el personatge i avaluar si aquest pot moure's o no cap allà. En cas que pugui el moviment es realitzarà tal i com s'havia fet fins al moment, i en cas que no el personatge no ha de respondre al moviment. Prèviament a efectuar el moviment es pot saber perfectament a quina posició anirà a parar el personatge, doncs sabent que la unitat mínima de moviment d'aquest és de 32 píxels, podem avançar que en intentar-lo moure aquest es desplaçarà 32 píxels en la direcció desitjada des de la seva posició actual. Així doncs, i aprofitant la matriu que s'ha utilitzat per pintar els obstacles, es pot calcular a partir de la posició (en píxels) del personatge quina és la posició de la matriu que l'hi correspon (dividint les seves posicions X i Y per 32). Si es pot aconseguir la seva posició actual, doncs, també es podrà aconseguir la seva posició futura (sabent que avançarà 32 píxels cap a una direcció concreta). Amb això s'haurà de limitar el moviment de forma que si en prémer una tecla la posició a la que caminaria el personatge conté un 1 en la matriu d'obstacles aquest moviment no es dugui a terme. Això farà que mentre el personatge es mogui cap a posicions que contenen un 0, un 2 o un 3 (o sigui, o res, o flors o pedres) el personatge es mogui normalment, mentre que si es dirigeix a una posició que conté un 1 el moviment no es dugui a terme i el personatge segueixi parat al mateix lloc des d'on es volia iniciar el moviment.

Amb això doncs neix la implementació del sistema de col·lisions que s'utilitzarà al llarg de tot el programa. La base de la part gràfica està creada, així com també l'inici del sistema de capes del mapa i el sistema de moviment i col·lisions del personatge.

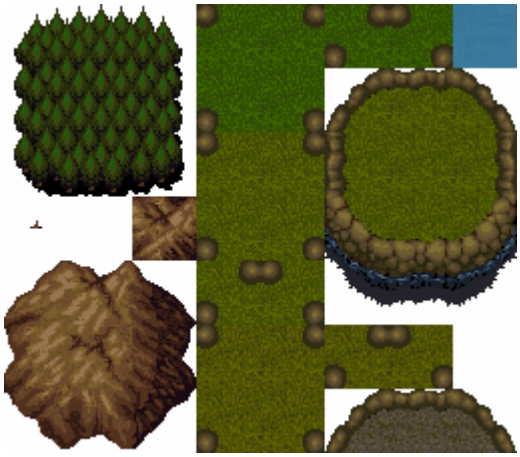
Implementació – 5: Aspecte gràfic

5.1: Aspecte gràfic: *tilesets*

Anteriorment ja s'ha comentat que els *tilesets* són unes imatges que representen un conjunt de *tiles* que poden ser terreny o obstacles, mentre que els *charsets* són imatges que representen un o varis personatges amb totes les seves posicions possibles. Malgrat els *tilesets* i *charsets* utilitzats fins el moment han anat més que bé per realitzar el treball fet, ara s'ha de començar a pensar en l'aspecte que definitivament haurà de tenir el joc. Això és una qüestió d'estètica i de preferències personals, així com de la "dimensió" que ha de tenir el joc. Amb això

de “dimensió” es fa referència a que, per la idea de joc en que es basa aquest projecte, els personatges i el terreny s’han de mostrar molt més petits (per tal de interpretar un torn com un moviment realment gran). Aquí doncs sorgeix la necessitat de buscar noves fonts gràfiques pel joc (o sigui nous *tilesets* i nous *charsets*). Després d’una llarga i infructuosa cerca per Internet, un servidor decideix intentar crear els seus propis *tilesets* i *charsets* a partir de captures de pantalla d’un joc ja existent. Això implica haver de capturar cada tipus de terreny i dividir-lo en una unitat repetible de 32 x 32, capturar cada obstacle i, a més de dividir-lo també, pintar de blanc tot el fons (per ser llavors interpretat com a transparència), i el mateix amb cada edifici i cada personatge.

Després de vàries hores perdudes en això l’esperança pot més que la paciència, i una nova cerca en profunditat per Internet treu a la llum un *tileset* que presenta un aspecte fantàstic i molt adient pel joc, així com una proporció igualment encertada. El nou *tileset* (de tamany realment monstruós, ni més ni menys que 256 x 16448 píxels agrupats en *tiles* de 32 x 32) presenta una sèrie de *tiles* perfectes per representar tant la superfície, com obstacles, com edificis (tenint ja en compte que en el futur joc hi hauran d’haver edificis). Un exemple del terreny:



Així com d'alguns obstacles (a part de les muntanyes i boscos apareguts anteriorment):

I alguns dels edificis que més endavant es podrien utilitzar per representar castells o mines, per exemple:

Aquests edificis comporten una sèrie de problemes: un d'ells és la seva disposició respecte el *tile* on es troben. El castell, per exemple, en la part de la muralla que està inclinada, és un tile on el personatge no hi hauria de poder caminar per tal de no enfilar-s'hi literalment, però el fet de no poder-hi anar fa que el personatge en quedi molt allunyat. Un altre dels problemes són les parts altes del edificis que pertanyen a *tiles* superiors. O sigui que, visualment, la part més alta de la torre del castell, tot i saber que no ocupa espai en el "terra", és una part de terreny on el personatge no pot anar. En cas de permetre que el personatge pugui caminar-hi el que s'aconsegueix és que el personatge camini per sobre la torre. L'única solució a això seria determinar quines *tiles* s'han de pintar abans del personatge (els que aquest hi passarà per davant) i quins després (els que quedaran per davant del personatge). Això en part també passa pel fet que el personatge no sigui una imatge de 32 x 32, sinó de 32 x 48. A part de portar tota aquesta sèrie de problemes (ja que una part queda per sobre d'un tile que no l'hi correspon), queda excessivament gran respecte la resta del terreny, de forma que sorgeix aquí la idea de buscar un nou *charset* amb un o varis personatges més petits. Abans, però, el problema dels edificis també va obliga a buscar si existeix algun *tileset* amb edificis que, encara que siguin més senzills, tinguin una mida de 32 x 32 estricte. La resposta arriba en forma d'un conjunt d'edificis que no només són perfectes per la seva mida i tamany, sinó que s'adapten perfectament als *tiles* de 32 x 32:

D'aquí doncs en poden sorgir perfectament 2 castells i 3 tipus de mines, per exemple. Un cop decidit l'aspecte del terreny on es desenvoluparà el joc, s'ha de buscar un *charset* que compleixi les expectatives estètiques i de dimensió esmentades anteriorment.

5.2: Aspecte gràfic: *charsets*

El personatge que representarà l'avatar en el joc ha de tenir un aspecte medieval (per tal i com s'ambienta el joc), respectar un tamany de 32 x 32, i presentar un aspecte (en tamany, colors i tipus de dibuix) que concordi amb el terreny utilitzat fins al moment. Després de molt buscar per la xarxa apareix un *charset* que no sols compleix aquestes expectatives, sinó que sembla provenir del mateix tipus de dibuix:

En aquest *charset* hi ha segurament tots els personatges que es necessitaran pel joc, presentant tots un mateix aspecte i una mida que concorda perfectament amb el terreny. L'únic problema que presenta és que, enlloc de 4 animacions com tenia el *charset* anterior, aquests personatges en tenen 3. No obstant, observant el primer s'aprecia que una de les 4 posicions és repetida (quan el personatge està amb les dues cames iguals, immòvil), de forma que l'únic que aquesta diferència implica és substituïr els moments en que el personatge pren la última posició per la segona posició d'aquests, i llavors tornar a iniciar normalment la seqüència de l'animació.

Amb aquests recursos ja es disposa del material suficient per crear un mapa amb un terreny, una sèrie d'obstacles, uns edifics, i uns personatges. Aquí sorgeix la necessitat (apareguda anteriorment com a idea) d'estructurar el mapa en una sèrie de capes de terreny.

5.3: Aspecte gràfic: capes del mapa

Si anteriorment ja s'havia dividit el mapa en dues capes pertanyents al terreny i als obstacles, res impedeix afegir una nova capa per a edificis, per exemple. No obstant l'existència d'un major nombre d'obstacles (així com d'edificis), i tenint en compte que les imatges es divideixen en *tiles* de 32 x 32, fa que es necessiti un major nombre de números per a indicar quin és el *tile* que s'ha de pintar a continuació. La inclusió d'una nova capa per a edificis independent de la capa d'obstacles (reservada per a obstacles del terreny) fa que es puguin repetir números per a referir a imatges diferents, amb posicions diferents dins aquestes. No obstant, aquests números no són suficients per a representar tots els *tiles* que poden aparèixer, així que desaparéixen els números per tal de donar pas a les lletres. Més de 20 lletres front a 10 números doblen el nombre de *tiles* als que es pot fer referència per una mateixa capa. Així doncs el mapa de moment disposa ja de 3 capes: una pel terreny, una pels obstacles, i una pels edificis.

Arribat aquest moment definir aquestes noves matrius com a variables estàtiques del programa implica limitar aquest a un sol mapa, així que en aquest punt neix el mapa com a un conjunt de fitxers de text: fitxers simples que contenen una cuadrícula de (en aquest primer cas) 32 x 24 caràcters, que en iniciar el programa cada un d'ells es carrega a la seva taula corresponent copiant exactament cada caràcter i que, arribat el moment de pintar-se, cada taula utilitza les seves propies relacions entre lletres i posicions dins una imatge (que també poden ser diferents per cada capa, simplement definint una nova textura pel nou *tileset*). Amb això doncs en cada execució el programa carrega els diversos fitxers del mapa a les seves matrius corresponents i, alhora de pintar-los, passa els diferents caràcters de cada posició per una sèrie de *switchs* enormes (que no tarden en passar a convertir-se en accions) per després considerar la possibilitat de tenir-los com a fitxers de text associats a la capa que pinten, de forma que en carregar la capa el programa crea també una taula formada per *structs* que contenen una lletra i les dues posicions de fila i columna a la que aquesta està associada amb el *tileset*. D'aquesta manera cada nou *tileset* que s'utilitza pot tenir la seva pròpia relació entre les lletres que el representen a la matriu del mapa i la posició que a cada una li pertoca alhora de pintar la capa. L'aspecte doncs que ofereix el joc en aquest moment és el següent:

Un cop arribat en aquest punt l'estructura de capes del mapa ha pres realment el seu sentit, els mapes es carreguen des de fitxer, i les seves relacions amb fitxers d'imatge també. Ara, és hora de començar a implementar el sistema de joc en sí.

Implementació – 6: Combats

6.1: Combats: aspecte

Una de les coses que ha de tenir aquest joc són combats i, per tant, cal dissenyar un sistema de batalles estructurat igualment per torns, amb un aspecte propi i un funcionament.

Les batalles per torns es poden estructurar de diferents maneres: una opció és que les diferents criatures que hi participin tinguin un atribut d'iniciativa, i que els seus torns es succeeixin des de la que té el valor major fins a la que el té menor, per després tornar a començar. Un altre sistema es basa en una *cooldown* (o refredament) després d'un torn, en el qual una criatura, després de realitzar la seva acció, ha d'esperar un temps determinat abans de poder tornar a actuar. Per últim, un sistema per torns es pot estructurar simplement per jugadors, de forma que primer un jugador realitza totes les accions que pot, per després donar pas a l'altre jugador.

Per aquest joc en particular el sistema escollit és l'últim i, tal i com s'estructura el sistema d'un avatar amb criatures, això es tradueix de forma que el primer jugador ataca amb totes les seves criatures, després el segon jugador fa el mateix, i així tornant a començar. L'avatar (com ja s'ha explicat en anterior apartats d'aquesta memòria) no combat directament a la batalla, sinó que actua sols com a

representació del jugador, o sigui representant qui a nivell del combat dona les ordres a les seves criatures i dirigeix els atacs, però sense prendre'n part. Això per tant deixa a la vista un sistema de combat amb dos grups de criatures enfrontats entre ells. Una opció comú pels jocs que utilitzen aquest sistema és la de presentar els dos grups de criatures encarades un a cada extrem de la pantalla. No obstant, un altre sistema utilitzat en menor grau però millor pel material disponible és el de presentar el grup de criatures a qui s'ataca directament a la pantalla, interpretant així que les criatures que ataquen estan "fora" d'aquesta, igual que el jugador. Això dóna l'efecte que els atacs es dirigeixen cap a la pantalla, i que provenen d'aquesta. Aquesta opció es pren també en gran part pel material que es troba referent a les criatures. Una sèrie d'imatges anomenades *battlers* que simplement representen una imatge frontal de la criatura i que, per tant, fan d'aquest sistema la opció més assequible. A continuació es presenten unes mostres d'aquests anomenats *battlers*:

Així doncs, decidit ja com serà el sistema de combats, i disposant del material necessari per a realitzar-lo, és necessari implementar aquest sistema.

Per a mostrar una batalla es decideix crear una nova finestra independent de la finestra on es mostra el mapa, que s'obrirà quan el jugador entri en una batalla i es tancarà un cop aquesta acabi. La nova finestra ha de tenir també una superfície on poder pintar les criatures, així com una sèrie de botons i components de Windows per poder escollir l'acció que durà a terme cada una d'aquestes, o per veure el resultat de cada acció. Així doncs, sabent que just al principi de la implementació d'aquest projecte s'ha assignat com a superfície per pintar una

finestra sencera, potser també es pot assignar com a superfície una part de la finestra delimitada per, per exemple, un *panel*. Així doncs, aquesta finestra es composa per un *panel* que ocupa la major part d'aquesta, però deixant espai suficient per incloure-hi en un futur la informació necessària sobre les criatures participants en el combat, així com les accions que aquestes realitzin o puguin realitzar.

Un cop creada aquesta nova superfície es necessita mostrar les criatures i, molt important, mostrar-les sobre quelcom més que el color gris per defecte de les finestres de Windows. Texturitzar el fons mitjançant *tiles* ara no té massa sentit, ja que enlloc de tenir una vista superior d'un terreny en un combat es presenta una vista en perspectiva i amb profunditat. Com en d'altres jocs que utilitzen aquest mateix sistema de combat, la millor opció per a mostrar un fons és una imatge en perspectiva del terreny corresponent, que es mostri com el punt de vista d'un individu allà present. Això dona sensació de profunditat i, a més, presenta una superfície representada com el terra de la imatge on les criatures poden dibuixar-se perfectament, aparentant que aquestes es troben realment d'empeus sobre el terra.

Després de molt buscar una imatge que serveixi com a fons (tenint en compte que ha de ser una imatge amb profunditat, amb perspectiva natural, amb un terreny no abrupte on les criatures puguin ser dibuixades naturalment, i sense que hi surti ningú saludant amb la mà, que malgrat semblar una cerca fàcil, és extremadament difícil trobar una imatge que compleixi aquests requeriments) és l'hora de dibuixar les criatures. Tenint en compte que el nombre que se'n mostri pot variar (ja que tant n'hi poden haver 3 com 1) cal trobar una raó que permeti dibuixar les criatures sense que es tapin unes a altres i sense que quedin descentrades. Donat que les batalles es mostren en una finestra a part i que, per tant, en aquesta finestra s'han de passar per paràmetre les dades que s'hagin d'utilitzar (com l'avatar amb totes les seves criatures o la matriu de criatures enemigues), es pot aconseguir d'aquestes dades el nombre d'enemics que hi ha. D'aquesta forma, dividint el tamany de la pantalla entre el nombre d'enemics + 1 es pot aconseguir (sense contar els extrems) tants talls com nombre d'enemics hi ha, cosa que després n'hi ha prou en dibuixar cada enemic en la posició del tall que l'hi pertoca, tenint en compte el tamany de la imatge de l'enemic (per tal de dibuixar-lo centrat al tall, i no a partir d'ell).

Un cop tot això està implementat el sistema de batalles presenta un aspecte més que satisfactori, mostrant un fons en perspectiva i un seguit de criatures de cara a la pantalla perfectament integrades en aquest fons:

6.2: Combats: sistema de control

El següent pas a dur a terme consisteix en implementar el sistema de les batalles, o sigui com es succeiran els torns, quines accions podran dur a terme les criatures de l'avatar, com aquestes influiran en les criatures enemigues i, molt important, com aquestes reaccionaran i atacaran, o sigui, el torn enemic.

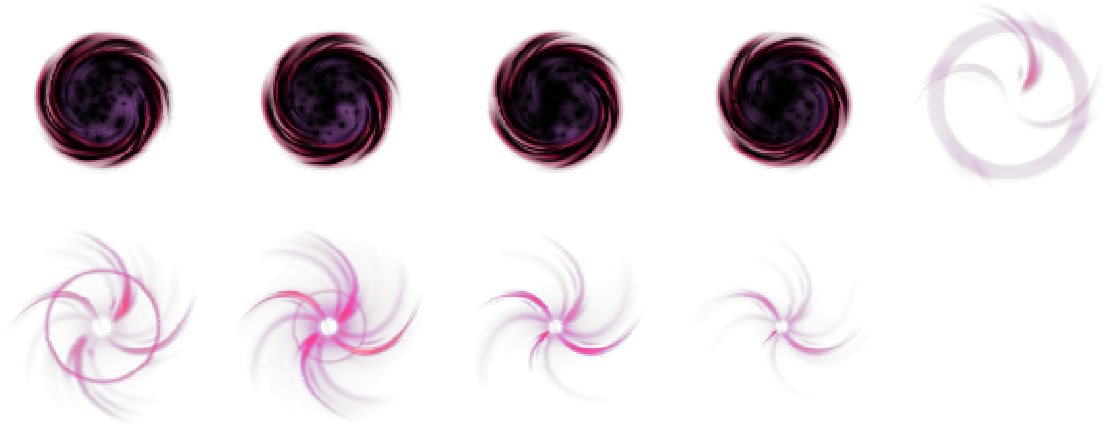
Donat que la superfície on es mostra la batalla és un *panel* i, per tant, queda superfície lliure a la finestra, es decideix que les criatures de l'avatar es mostrin en aquesta espai lliure de la finestra, així com les seves dades i les accions que aquestes puguin dur a terme. Primerament sorgeix un problema: quantes criatures es mostraran? Quantes en té l'avatar a la seva disposició? Si hi ha lloc per 3 i només se'n mostra 1, què passarà amb l'espai desaprofitat? La solució a aquestes preguntes arriba en forma d'una sola posició per mostrar una criatura, i una llista de criatures, de forma que es mostra sempre el nom de totes elles, però només es mostra la informació de la criatura que en aquell moment tingui el torn. Així doncs, durant el torn les criatures es van succeïn una rere l'altre, i en canviar de criatura s'ha de modificar també tota la informació que se'n mostri. Just iniciar la batalla es composa una llista amb tots els noms de les criatures que formen part de l'avatar i

es mostra la informació provinent de l'objecte criatura de la primera d'elles. El botó que correspon a atacar en aquell moment només serveix per fer avançar la posició de la llista a la següent criatura i fer així també que la informació mostrada es refresqui amb les dades de la següent criatura. Problema: diferents criatures tenen diferents imatges per mostrar, cosa que implica que un *pictureBox* (quadre d'imatge) de tamany fix farà que algunes imatges apareguin incompletes, mentre que un de tamany variable farà que per algunes imatges quedi molt petit, mentre que per d'altres prengui un tamany descomunal arribant fins i tot a sobresortir per sota els botons o la llista de criatures. Això planteja un nou problema que troba solució en un altre tipus de *set* diferent als vistos anteriorment i que, en un principi, només s'havia d'utilitzar per l'avatar: els anomenats *facesets*. Igual que els *tilesets* o *charsets*, els *facesets* són imatges formades per un conjunt d'imatges de menor tamany però que, enlloc de representar terreny o personatges, representen cares.

D'aquest seguit de *facesets* que es troben i que, afortunadament, corresponen als personatges del *charset* i a algun dels *battlers* utilitzats es treu la solució al problema de les imatges: mostrar la cara de la criatura atacant, enlloc de la seva imatge. Això, a més, diferencia molt millor quina és la criatura que ataca i quina la que és atacada.

Un cop solucionat aquest problema, s'ha d'implementar d'alguna manera el funcionament dels atacs que les criatures de l'avatar duran a terme cap a les criatures enemigues. Si s'ha fet servir una llista per les criatures de l'avatar, sembla una bona opció utilitzar també una llista on poder seleccionar les criatures enemigues i mostrar-ne els seus atributs. Així doncs, i tenint en compte que l'atac s'haurà de dirigir cap a una criatura, en tenir-ne alguna seleccionada i prémer el

botó atac s'inicia una acció que calcula els punts de mal que l'atac ha de causar (en funció de la criatura atacant i del seu nombre, així com un petit coeficient aleatori) i que llança aquest mal contra la criatura seleccionada. L'atac en aquest punt ja funciona, però es presenta com quelcom instantani i quasi inapreciable. No obstant, entre tal quantitat de *sets* hauria set estrany que no hi hagués hagut algun *set* destinat a emmagatzemar animacions d'atac i, efectivament, aquests sets existeixen.



Aquests *sets* (anomenats simplement *animations*) inclouen tot el seguit d'animacions que un atac mostrarà. Així doncs, d'igual manera que amb el moviment de l'avatar pel mapa, en atacar s'activa un booleà que mostra, centrat a la posició de la criatura que rep l'atac, tota l'animació de principi a fi, juntament amb la informació associada a l'atac (com la criatura que l'ha dut a terme, a qui ha atacat, i els punts de mal que ha causat). Arribat aquí doncs, és l'hora de permetre a les criatures enemigues defensar-se.

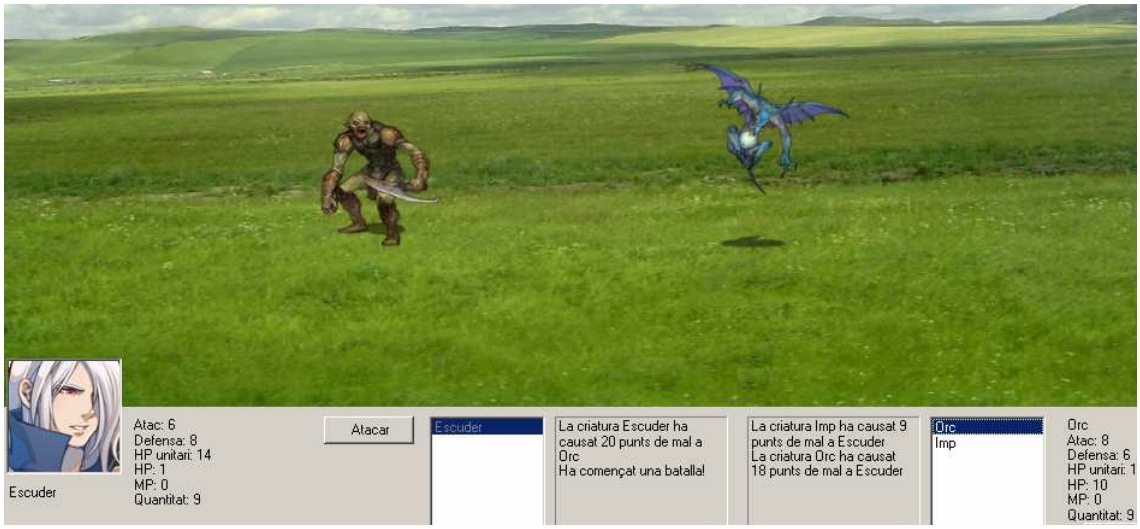
Un cop totes les criatures de l'avatar han atacat cal començar el torn del contrari i, per tant, permetre que totes les criatures rivals ataquin. El sistema que aquestes hauran d'utilitzar és el mateix que el que utilitzen les criatures de l'avatar però, evidentment, amb un alt component aleatori. D'una en una, seleccionen alteatòriament una criatura de les que acompanya l'avatar i duen a terme aleatòriament un dels seus atacs disponibles (després d'haver fet, si és necessari, la comprobació pertinent, com ara comprobar que es disposa dels punts de màgia necessaris si es vol fer un encanteri). Amb això es calcula de nou els punts de mal que la criatura enemiga causarà i aquests s'apliquen a la criatura que els ha de

rebre, mostrant per pantalla igualment la informació referent a l'atacant, l'atacat i la quantitat. Un cop criatura rere un altre duen a terme el seu atac, es repren el torn de l'avatar, i torna a començar la seqüència. Falten però dos detalls sumament importants: la mort i el final de la batalla.

Una criatura es considera morta quan la seva quantitat arriba a 0 (ja que, com s'ha explicat anteriorment, les criatures no són tal sino grups d'aquestes). Així que, si la quantitat d'una criatura arriba a 0 o per sota la criatura és considerada morta i es treu de les llistes de forma que ni el jugador la pot controlar, ni la criatura pot ser atacada (o atacar en cas de ser controlada per la màquina). Així doncs per la part pertinent al jugador això només implica la necessitat de, quan una criatura mor, treure-la de la llista on es trobi (d'aquesta forma ni el jugador en pren el control ni el jugador la pot seleccionar com a objectiu del seu atac). Per part de la màquina, no obstant, obliga a comprobar que la criatura que controla no estigui morta abans de fer res, i a comprobar que la criatura a qui atacarà no estigui ja morta.

El final de la batalla, doncs, es deriva directament d'aquí, i és que després de cada torn es llença una acció que comproba si totes les criatures d'un bàndol estan mortes i, en cas afirmatiu, finalitza la batalla i mostra el guanyador per pantalla.

En aquest punt, doncs, les batalles contra la màquina queden perfectament implementades, i presenten el següent aspecte:



Implementació – 7: Sistema de torns

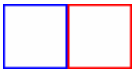
Tenint ja les batalles operatives i el mapa construït, és hora de tirar enrere fins als principis més fonamentals del joc i recuperar el concepte: joc per torns. Malgrat el sistema de batalles ja està estructurat per torns, és necessari ara estructurar el joc per torns fora de les batalles, a la pròpia pantalla del mapa. No és un sistema que tingui molt sentit per una partida individual, però sí que pren sentit quan hi ha més d'un jugador (ja sigui humà o controlat per la màquina), o sí que pren sentit quan es pensa en el guany de recursos o el repoblament de criatures (ambdúes coses s'implementaran més endavant).

Així doncs és necessari limitar el moviment de l'avatar i dividir-lo en una sèrie de torns, establint un moviment màxim per torn i una acció que permeti passar de torn. Ja que amb el sistema de batalles s'ha dividit la finestra en un *panel* on mostrar l'acció i una part de la finestra disponible per mostrar informació o botons, també es pot fer el mateix amb la finestra del mapa, de tal forma que queda una zona disponible per mostrar informació referent a l'avatar i al torn (així com per a recursos en un futur). Aquest torn, doncs, es pot fonamentar en 10 accions per torn, entenent per acció qualsevol moviment o interacció amb events (també de cara a un futur). Aquí és on es veu modificat l'atribut moviment de l'avatar amb cada moviment fructuós que realitza (doncs si es modifica amb cada tecla l'avatar restarà moviment encara que es trobi davant d'unes muntanyes i intenti avançar sense èxit) de tal forma que, en arribar a 0, l'avatar queda incapaç de moure's fins que el jugador no prem el botó de passar torn, moment en què es recupera tot el moviment de l'avatar. Com ja s'ha comentat anteriorment, aquest pas de torn servirà també per augmentar els recursos dels que disposarà l'avatar.

Implementació – 8: Mines i recursos

Ja s'ha parlat anteriorment d'un sistema de recursos del joc que funcionarà a través d'uns components que es guanyaran cada torn en funció de la quantitat de mines que el jugador tingui al seu poder, excepte l'or (o l'equivalent als diners) que augmentarà una quantitat fixa cada dia. Per tant i sabent això és necessari que repartides pel mapa hi hagi una sèrie de mines on l'avatar es pugui dirigir per tal d'apoderar-se'n, Aquestes mines es representen com uns edificis (de menor tamany que el castell) i es troben disperses pel mapa. Quan un avatar s'hi acosta i hi interactua la mina passa a ser possessió del jugador (quedant marcada amb el seu color) i, mentre romangui essent seva, proporciona una quantitat de recursos cada dia (o sigui cada torn). La representació gràfica de les mines és doncs un dels edificis que s'han vist en apartats anteriors d'aquest document, i la interacció que hi tindrà l'avatar és molt simple. Si en aquella casella, en una nova capa que servirà per marcar les possessions, no hi ha la lletra que representa el color de l'avatar (pot no haver-n'hi cap o haver-hi la lletra que representa el color contrari), aquest pot interactuar amb la mina de forma que gasta un punt d'acció, suma una mina d'aquell tipus a la quantitat de les que en disposa, i aquella mina queda marcada del seu color. Si la mina ja és del color de l'avatar aleshores aquest no hi podrà interactuar, doncs sino passaria que l'avatar podria anar perdent punts d'acció amb una mina amb la qual no pot interactuar perque ja és seva. A partir de llavors cada torn que passa el jugador (i mentre la mina segueixi essent seva) li proporciona una quantitat determinada d'un recurs concret. Per a les marques de

colors i un cop assimilat el funcionament dels sistemes de *sets* es crea un petit *set* que simplement conté les marques dels colors a qui pot pertànyer una mina:



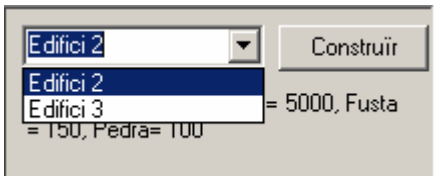
Aquestes marques doncs es pintaran recorrent una nova matriu de possessions que guardarà a la mateixa casella que la situació de la mina la lletra corresponent al color del jugador que la posseeixi. Un cop implementat aquest sistema i en funcionament el resultat que es mostra per pantalla segons si una mina és o no propietat del jugador és el següent (a part que, un cop implementat el sistema de recursos i mines, aquesta informació es mostra també a la pantalla, sota la informació de l'avatar):

Implementació – 9: Castells

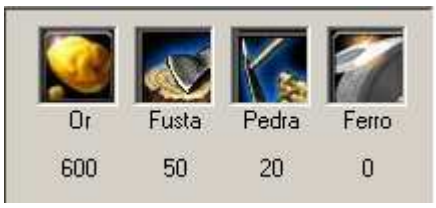
L'últim element imprescindible per a acabar l'estructura del jugador és un castell. El castell com ja s'ha detallat anteriorment és el lloc on el jugador podrà (si disposa de recursos suficients) construïr nous edificis, reclutar criatures, moure criatures de la defensa a l'avatar o viceversa, etc. Per tal de poder-lo mostrar el castell s'estructurarà en una nova finestra (igual que els combats, per exemple), però on no caldrà que hi hagi tot el sistema per pintar la pantalla ja que la imatge del castell serà fixa. La idea inicial és un fons que representa el castell, i un seguit de *panels* i botons que permeten realitzar totes les accions necessàries dins el castell. La imatge de fons, per exemple, pot ser perfectament aquesta:

I és sobre d'aquesta imatge on es col·loquen els components necessaris per interactuar amb el castell. N'hi ha 5 d'imprescindibles: Un bloc on es puden construïr nous edificis (que inclouen tots els edificis que es poden construïr, així com el seu cost), un bloc on es mostren els recursos dels que disposa el jugador en aquell moment, un bloc on es poden reclutar criatures (on es mostren els seus atributs, la seva quantitat disponible, etc.), un on es mostren les defenses que actualment hi ha al castell (en cas d'haver-ni), i un per mostrar l'avatar i les seves criatures (en cas que l'avatar sigui al castell en aquell moment). També s'ha de tenir en compte que entre l'avatar i les defenses ha de ser possible moure criatures, i que les criatures iguals s'agrupen enlloc d'ocupar vàries posicions.

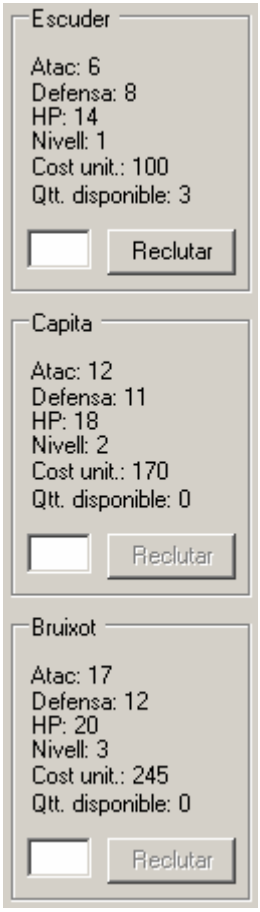
Primer de tot de cada un dels edificis construïbles (que es despleguen en un *listBox*) s'ha de mostrar el seu cost i permetre l'opció de construïr-los en cas de disposar de suficients recursos i de que no estiguin construïts ja. L'objecte jugador té com a atributs un seguit de booleans que indiquen si els edificis en qüestió estan construïts, així com uns mètodes que permeten construïr els edificis a canvi dels recursos indicats. Així doncs, aquest bloc es mostra de la següent forma:



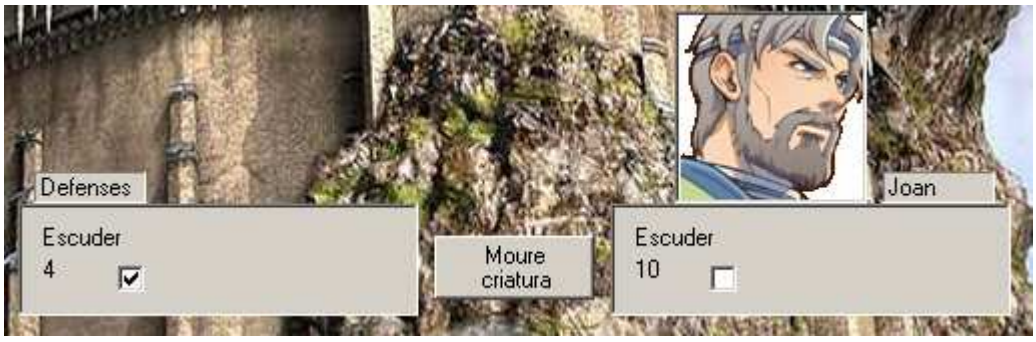
El bloc destinat a mostrar recursos no té més complicació que la de prendre el nom, la imatge i la quantitat de recursos del jugador i mostrar-los per pantalla, només tenint en compte de refrescar la informació després d'haver construït quelcom o haver reclutat criatures:



Pel que fa al bloc que ha de permetre reclutar criatures, s'ha de tenir en compte primerament que per defecte només es pot reclutar un tipus de criatura en concret, ja que per a poder reclutar els altres tipus de criatura és estrictament necessari haver construït abans els edificis pertinents. Per a estalviar comprobacions abans de reclutar la criatura, es desactivarà el botó que permeti reclutar una determinada criatura en cas que el seu edifici corresponent no estigui construït. Si la criatura és reclutable només s'ha de tenir en compte la quantitat disponible que n'hi ha, la quantitat sol·licitada, i el cost derivat de reclutar aquesta determinada quantitat de criatures. En cas que tot això es compleixi correctament, es procedeix a reclutar el nombre de criatures determinat (creant un nou objecte en cas que la criatura no existeixi encara, o sumant la quantitat de criatures determinada en cas que es trobi a la defensa aquella criatura en concret). El *panel* per reclutar criatures presenta el següent aspecte:



Evidentment és necessari disposar d'un bloc que mostri les defenses del castell i que permeti afegir-ne o treure'n. Aquest bloc va íntimament lligat amb el bloc corresponent a l'avatar ja que sí aquest es troba al castell es poden treure criatures de les defenses per posar-les a l'avatar o viceversa. En el bloc de les defenses es mostra el nom i la quantitat de criatures, i en el bloc de l'avatar es mostra el nom de l'avatar, el seu rostre, i les criatures que l'acompanyen. Cada criatura (tant de l'avatar com de la defensa) té associat un *checkBox* que, en marcar-lo, fa aparèixer un botó que serveix per moure criatures d'un cantó a l'altre, recordant que les criatures del mateix tipus d'agrupen automàticament i les criatures de tipus diferents necessiten un nou lloc a les defenses (o a l'avatar) per poder-les posar. L'aspecte que mostren ambdós blocs conjuntament és el següent:



Així doncs fins aquí el disseny i implementació del castell, que és un element de vital importància per al joc. Serà també aquí on l'avatar es recuperi un cop mort, però això s'explicarà en un apartat posterior de la implementació.

Implementació – 10: Events i esdeveniments

Fins aquí s'ha parlat de diversos esdeveniments que poden tenir lloc en un mapa tal com un combat, una mina, o un castell. No obstant, fins ara no s'ha explicat com funciona el sistema per fer que el programa detecti quan l'avatar entra en contacte amb un d'aquests esdeveniments, i com ho fa per iniciar-lo. L'encarregat d'això és el sistema de capes en què es divideix el mapa.

Fins aquí les capes de les que disposa el mapa són la capa de terreny (que serveix de base), la capa d'obstacles, la capa de camins (que és una capa intermitja de la qual no s'ha parlat i que podria estar perfectament inclosa en la capa d'obstacles, però separar-la facilita la seva traducció entre lletres i posicions), la capa d'edificis i, finalment, la capa d'events o esdeveniments, juntament amb una capa de terreny que es genera a partir de les anteriors i que, simplement, marca com a no passables qualsevol casella de la capa d'obstacles i d'edificis que tingui alguna lletra per valor, facilitant així el procés de la detecció de col·lisions amb una sola matriu. La capa d'esdeveniments funciona de manera molt semblant a aquesta, doncs parteix del principi que qualsevol edifici present en el mapa serà un event (un castell o una mina) i directament marca la posició corresponent de la seva capa amb una "e". L'avatar, en moure's pel mapa, si topa amb una "e", crida automàticament una funció que s'encarrega de comprobar de quin tipus d'esdeveniment es tracta. Sabent la posició on es troba aquesta "e" el programa obre un fitxer de text destinat a emmagatzemar totes les coordenades on hi ha

esdeveniments junt amb les seves dades. El què fa el programa és buscar en aquest fitxer les coordenades corresponents i, quan les troba, llança un *switch* per veure de quin esdeveniment es tracta (un castell, una mina o un combat), per després seguir llegint i carregar les dades necessàries per executar l'esdeveniment (com de quin castell es tracta, de quin tipus de recursos és la mina, o quins enemics hi ha a la batalla, així com quin fons es mostra, i quina música hi sona). Val a dir que per les batalles amb enemics controlats per la màquina existeix una altra capa que mostra una imatge (provinent d'un *charset* de criatures) de la criatura (o d'una de les criatures) que hi haurà a la batalla i que s'interpreta en generar la capa d'esdeveniments com una "e" també.

Així doncs aquesta capa és la que s'encarrega de detectar un event i llançar-lo segons sigui d'un tipus o un altre.

Implementació – 11: Mort i ressurrecció

És una realitat que, d'entre totes les batalles en les que l'avatar lluitarà, algunes les guanyarà i d'altres les perdrà. Les criatures mortes desapareixen un cop finalitzada la batalla, però què passarà quan totes les criatures de l'avatar hagin mort? L'avatar en sí, com a personatge, no morirà mai, simplement quan perdi una batalla l'avatar representarà que queda sense exèrcit i fuig de forma que el jugador podrà tornar a reclutar-lo (pagant també una determinada quantitat). Així doncs quan el jugador perd una batalla el seu avatar desapareix i, si vol reclutar-lo de nou, ha d'entrar al castell i allà apareixerà un nou bloc amb la imatge de l'avatar, així com el preu de reclutament, i sempre que disposi dels recursos necessaris i que en tingui la voluntat pot reclutar de nou l'avatar, que reapareixerà a les portes del castell. El nou bloc implementat al castell que permetrà reclutar l'avatar un cop mort presenta el següent aspecte:

Implementació – 12: Batalles no repetibles

Aquests esdeveniments als que s'ha fet referència abans i que es tracten com si fóssin edificis presenten arribat aquest punt un problema: la seva persistència és la mateixa que la d'un edifici qualsevol. O sigui que, si l'avatar es dirigeix al combat i el guanya, les criatures romanen allà de nou, de forma que l'avatar hi pot tornar a lluitar sempre que vulgui (cosa a la llarga molt infructuosa). Per tant, d'alguna manera o altra s'ha de crear un sistema per fer que, si l'avatar guanya aquestes batalles, els enemics desapareixin.

La primera opció pot ser crear una taula que emmagatzemi coordenades de tal forma que, abans d'iniciar una batalla, es busqui en aquesta taula si aquesta batalla ja s'ha realitzat anteriorment (entenent que en aquesta taula s'hi guardaran les coordenades de les batalles guanyades). No obstant, això suposa una alta càrrega afegint una cerca extra en una taula que en partides grans pot esdevenir realment llarga. Per tant, i sabent que la representació d'aquestes criatures es troba en una taula independent i que la batalla s'activa gràcies a l'esdeveniment que hi ha associat en la corresponent taula, n'hi ha prou en, si es guanya la batalla, buscar la posició corresponent a aquesta a la taula que representa les criatures i borrar-ne al caràcter, i el mateix per la taula d'esdeveniments, de tal forma que, un cop guanyada la batalla, en aquella posició la taula d'esdeveniments no en conté cap.

Implementació – 13: Final de partida

Arribat en aquest punt tota la partida individual està muntada i funciona correctament. Tant el mapa, com els combats, com el castell, com el sistema de

mines i recursos. Falta un petit detall que és la clau de qualsevol partida: el final. D'alguna manera o altra és necessari determinar un final per la partida, un moment en el qual es marqui un guanyador. Aquest punt, com ja s'ha comentat anteriorment, es marcarà per la pèrdua del castell (cosa que prendrà molt més sentit en implementar el mode de joc per 2 jugadors o contra la màquina). Així doncs en la definició de l'esdeveniment del castell falta incloure-hi una petita clàsula: si no hi ha defenses (o aquestes es vencen de forma que queden buides), la partida acaba. Aquest fi partida fa apréixer un nou botó a la pantalla indicant que la partida s'ha acabat i que tanca la finestra i, per tant, tota la partida carregada en aquesta.

Implementació – 14: Mode dos jugadors

Aquí comença la implementació del mode on de veritat el joc pren tota la seva dimensió: el mode dos jugadors. Aquest es desarrolla en un mapa on s'enfronten dos jugadors, els dos posseïdors d'un castell i un avatar, i igualment capaços de moure's i interactuar amb el mapa.

Donat que tot el sistema de moviment pel mapa, d'interacció amb mines, amb el castell propi o amb criatures del mapa ja està perfectament implementat, aquest ja serveix des d'un principi pel segon jugador. Aquí és necessari però implementar que, quan es produeixi el canvi de torn, prengui el control el jugador contrari i, per tant, les tecles i qualsevol interacció amb la interfície o el mapa les realitzi el jugador contrari. Així doncs a través d'un enter el programa identifica en rebre ordre de moure l'avatar o en executar un event a quin jugador toca i així actua fins que al passar torn torna a prendre el control el primer jugador.

Les diferècies entre aquest nou mode i el mode anterior prenen la seva importància en els enfrontaments: enfrontaments entre els avatars i enfrontaments entre un avatar i les defenses d'un castell. El problema que té el sistema de batalles implementat fins al moment és que en acabar el torn de l'avatar es llança automàticament el torn de l'enemic i aquest ataca aleatòriament, a part que la posició de les criatures sempre mostra el punt de vista des de l'avatar que ha iniciat l'atac i, per tant, només aquestes criatures es troben situades en la part d'interfície que en permet el seu control. El que s'ha de fer, doncs, és crear un nou tipus de batalla partint del primer on, després de cada torn, les posicions de les

criatures s'inverteixin i es passi així a mostrar frontalment les criatures que en aquest torn havien atacat, i a situar en la interfície controlable les criatures que han d'atacar a continuació. D'aquesta forma el control sempre es du a terme des dels mateixos components, i el jugador que ataca sempre veu la cara de la criatura que controla en aquell moment, i tota la imatge de les criatures que pot atacar. D'aquesta forma el sistema de batalles queda arreglat, i només falta detectar quan un avatar s'aproxima a un altre (doncs la capa d'esdeveniments aquí no serveix per res). Aquesta detecció es fa simplement comparant la posició on anirà a parar l'avatar que mou amb la posició on està l'altre avatar, i si aquestes coincidien es llança una batalla entre els dos avatars.

El sistema de batalles entre un avatar i les defenses del castell contrari és el següent punt a implementar. No obstant, un cop implementat el sistema anterior de combat entre 2 avatars aquest nou sistema no comporta gaire canvis, excepte que enlloc de l'avatar combaten les criatures a la defensa, i que la imatge que es mostra de fons ha de canviar per tal de reflectir que la batalla es du a terme a les portes d'un castell. Aquesta imatge doncs s'inclou com a atribut de la classe castell (de forma que cada castell pot tenir la seva pròpia imatge no només de dins el castell, sinó de les portes on es realitzin les batalles).

Implementació – 15: Mapa de mida superior a 32 x 24

Fins ara tot el joc s'ha dissenyat, basat i provat en un mapa inicial creat en un fitxer de 32 x 24 caracters que correspon per tant a un mapa de 32 x 24 *tiles*, que és la mida a la que queda dividida la pantalla prenent la mida del *tile* com a 32 x 32. Ara bé, un mapa no té perquè tenir aquesta mida màxima, i és que pot ser més gran. Que passarà quan un mapa sigui de mida superior? Ara per ara, per delimitar el moviment del personatge (i evitar així que surti dels limits del mapa), simplement es comprova abans de realitzar un moviment que la posició a la que es vol moure no surti del rang de la matriu on comproba el tipus de terreny (evitant també així l'error d'índex fora del rang). Però ara el mapa ha de poder tenir una dimensió major així que, a part de seguir tenint en compte que el personatge no surti del rang del mapa, s'ha de dissenyar algun sistema per tal que el personatge es mostri sempre a la pantalla i no ens quedi fora d'aquesta.

Aquest sistema parteix del primer bucle que s'ha fet en aquesta implementació: el bucle que recórre tota la pantalla de punta a punta pintant els tiles. Si enlloc de pintar la pantalla des de la posició 0,0 fins al final es pinta des d'un altre punt, la part del mapa que apareix per pantalla (en cas que sigui un mapa més gran de 32 x 24) és la que va des d'aquell punt fins a 32 *tiles* a la dreta i 24 avall. S'ha de tenir en compte però que s'ha de vigilar que a partir d'aquest punt existeixin el nombre de posicions que s'haurà de pintar o, en cas contrari, s'haurà d'agafar la posició més ajustada però que no surti del rang. Aquest punt a partir del qual pintarà s'ha de prendre en relació del personatge (i per tant ha de canviar cada torn), i s'intentarà sempre que sigui possible agafar una posició que faci que el personatge quedi al centre de la pantalla. Així doncs s'agafarà la posició X del personatge menys la meitat del tamany que es pintarà (que són 32 *tiles* per amplada) i el mateix amb la posició Y (però aquesta vegada amb 24 *tiles*). Abans de pintar, però, es s'ha de comprobar que aquesta posició no sigui en cap dels seus eixos inferior a 0, ni tampoc que sumant-hi el nombre de posicions que es pintaran superi el tamany de la matriu del mapa. Un cop comprobat això, doncs, a cada inici de torn el personatge es mostra centrat a la pantalla, o bé la pantalla per algun dels seus extrems (o per més d'un) mostra el límit del mapa, i això tant amb un sol jugador com en varis, només tenint en compte que si hi ha dos jugadors a cada torn la pantalla s'ha de centrar al jugador que toca en aquell torn.

Implementació – 16: Mode de joc contra la màquina

L'últim mode de joc que falta per crear és el mode contra la màquina. La idea d'aquest mode és que el jugador s'enfronti a un altre jugador igual a ell en tot, però controlat per la màquina. Així, aquest últim ha de tenir un avatar i un castell, i ha de permetre que l'avatar es desplaci a través del mapa, que s'apoderi de mines, que ataqui l'avatar del jugador, o que ataqui el castell del jugador. Així mateix ha de posseïr un castell propi on ha de poder construïr edificis i reclutar criatures. El mode del qual es partirà és doncs el mode dos jugadors però que, en passar torn, pren control la màquina.

El primer que ha de poder fer la màquina és moure el seu avatar. Es mourà tenint en compte les restriccions típiques del moviment de l'avatar, sense poder caminar per sobre obstacles i edificis, consumint un punt d'acció només per cada moviment executat amb èxit, i podent interactuar amb events (tot i que això es desarrollarà posteriorment). Aquest primer moviment doncs s'executa aleatòriament de forma que es mou fins que esgoti les seves capacitats de moviment. Així doncs un nombre aleatori determina la direcció en la que l'enemic s'ha de moure.

Segonament l'enemic ha de poder combatre contra el jugador contrari, contra les defenses del seu castell, o defensar-se contra l'atac de l'avatar. Aquí el sistema de combat que s'utilitza és el mateix utilitzat per combatre directament contra la màquina, només que passant com a array d'enemics les criatures que acompanyen l'avatar. Igualment passa en cas que el jugador ataqui el seu castell, o en cas que l'enemic ataqui el castell del jugador (serà una batalla entre les defenses del castell i les criatures de l'enemic). En cas que l'avatar contrari mori en aquestes batalles haurà de tornar-lo a reclutar en el castell, tal i com s'explicarà a continuació.

L'enemic ha de poder construïr, reclutar criatures i reclutar l'avatar en el seu castell. Al final de cada torn s'inicia la fase del castell en la qual, si disposa de recursos suficients, l'enemic pot recluta el seu avatar en cas que hagi caigut en combat, i seguidament iniciar la fase de construcció. Si disposa de recursos suficients i algun dels edificis no està construït encara el construirà, i un cop acabada la fase de construcció iniciarà el reclutament de criatures. Amb una probabilitat determinada (i sempre que dispongui dels recursos necessaris) el jugador controlat per la màquina reclutarà la quantitat màxima de criatures que l'hi

permetin els seus recursos o la disponibilitat d'aquestes, de forma que quedaran a la defensa per protegir el castell d'un possible atac del jugador humà.

Si bé anteriorment s'ha explicat com reaccionarà el jugador màquina en el transcurs d'una batalla, també és veritat que, a no ser que sigui atacat pel jugador humà, no té cap forma de participar en una batalla si no és a través d'esdeveniments. Així doncs en cada un dels moviments que l'avatar-màquina dugui a terme es s'ha de comprobar primer de tot si es mou cap a l'avatar humà (en cas afirmatiu, s'inicia la batalla corresponent entre els dos avatars), i seguidament s'ha de comprobar si la posició a la que es dirigeix conté un esdeveniment. En cas afirmatiu, comprobar de quin tipus d'esdeveniment es tracta i, en cas que no sigui batalla, atacar el castell del jugador humà o prendre la mina (en cas que no sigui ja seva). Igual que amb l'avatar jugador, moure cap a una mina pròpia no tindrà cap efecte i no restarà punts d'acció. Les batalles contra criatures del mapa són ignorades primer de tot perquè aquestes es plantegen com un repte pel jugador humà, i en segon lloc perquè avaluar si la batalla és factible abans de dirigir-s'hi (per evitar una conducta potencialment suicida) i desenvolupar la batalla sense interacció del jugador pot ser una clara avantatge pel jugador humà (deixant que l'oponent es vagi debilitant matant les criatures que troba pel pas sense raó de cap tipus).

Així doncs aquest mode permet a un jugador enfrontar-se a un oponent igual (o quasi igual) en possibilitats, i que desenvolupa una partida d'igual estructura que una partida de dos jugadors i, fins i tot, en els mateixos mapes que aquesta.

Implementació – 17: Música

L'apartat gràfic del joc funciona correctament, el sistema de joc està muntat i funcionen tots els seus modes. Què falta? A part dels menús (que es crearan seguidament), la música.

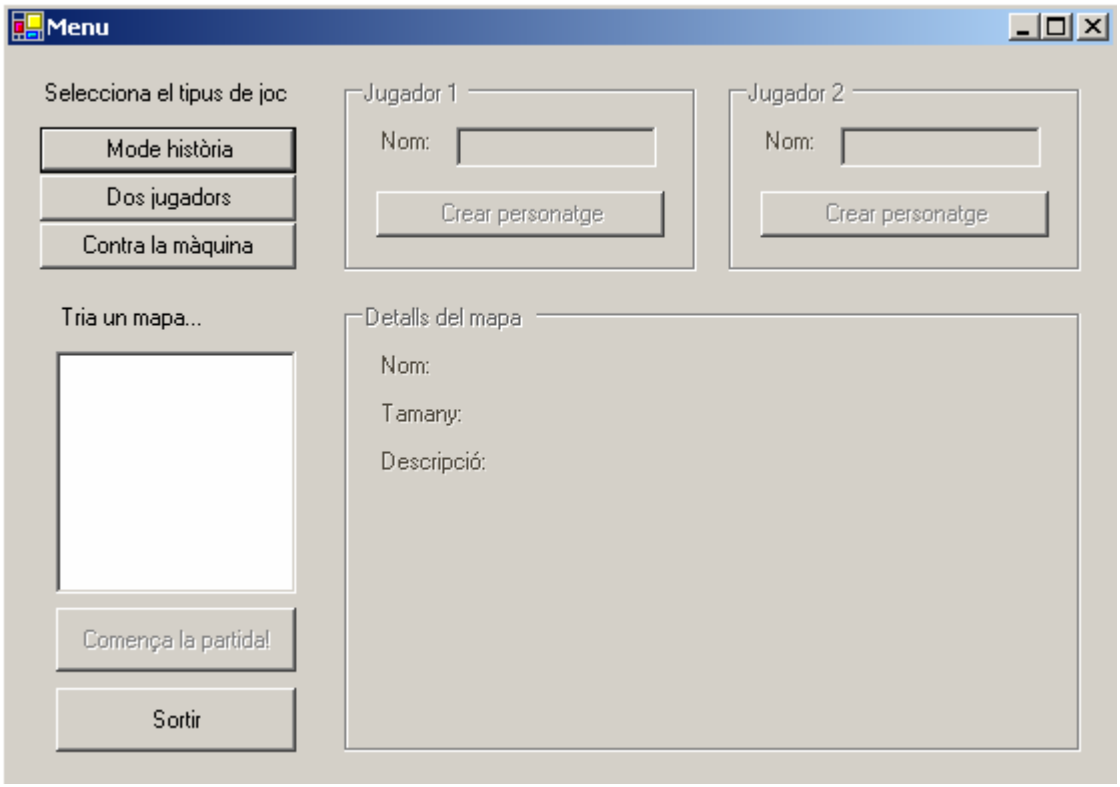
Per poder utilitzar la música en un projecte és necessàri utilitzar la llibreria AudioVideoPlayback continguda dins l'SDK del DirectX. Aquesta llibreria permet d'una forma fàcil i intuïtiva reproduïr arxius de música (i pel que porta a pensar el nom, suposo que també de video). Així doncs es pot declarar una variable Audio

en cada un dels forms que es vulgui utilitzar música, i assignar-li un arxiu .MP3 que es reprodueixi en rebre l'ordre Play.

La música és un component important dels jocs, doncs els dóna bona part d'ambientació, i accentúa el caràcter d'algunes escenes en concret. Així, qualsevol combat necessita una música adequada que presenti un cert grau d'emoció i risc, mentre que per exemple la música del castell humà ha de ser una peça gran i potent, a diferència del castell dels no-morts, que ha de ser una música lúgubre i tenebrosa. Per aquest apartat doncs s'han tret les músiques d'altres jocs ja existents.

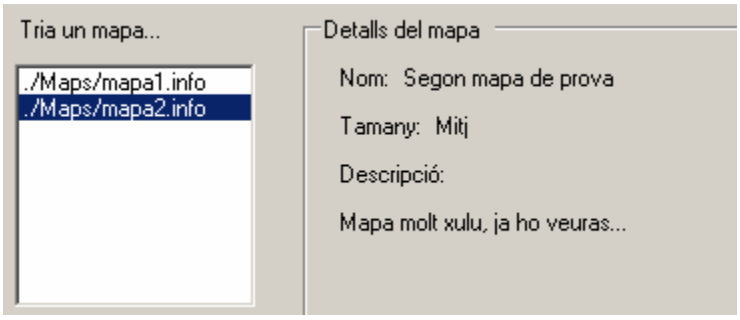
Implementació – 18: Menú per iniciar una partida

Amb aquest últim apartat de música el joc queda finalment estructurat i tancat, de forma que és possible iniciar una partida en qualsevol dels tres modes de joc i aquesta funciona perfectament. El proper pas és crear un menú on el/s jugador/s pugui/n escollir un tipus de partida, un nom pel seu jugador, pugui crear un avatar (triant el seu nom i el seu aspecte), i pugui triar un mapa. L'estructura pensada per a tal funció és una nova finestra de Windows amb 3 botons (un per cada tipus de joc), 3 *groupBox*, 1 *listBox* i 1 altre botó. D'aquests 3 *groupBox* n'hi ha un que permet introduïr el nom del jugador i crear un avatar (cosa que obre una nova finestra), un altre que és igual però per al jugador 2, i un que mostra la informació del mapa seleccionat (nom, tamany i descripció). Aquest mapa es selecciona des del *listBox*. Per últim, el botó serveix per iniciar la partida un cop tot està a punt.



Just en iniciar el menú tots els *groupBox*, el *listBox* i el botó per iniciar la partida estan desactivats, i només s'activen els necessaris quan el jugador selecciona un tipus de joc (pel mode història, per exemple, no s'activarà el *groupBox* que contingui la informació referent al jugador 2). Cada jugador ha d'introduïr un nom i crear un avatar. Aquest avatar ha de tenir un nom i un aspecte a triar entre una sèrie de perfils disponibles. Aquests perfils s'obtenen des d'un fitxer que conté el nom genèric d'aquests i el fitxer amb la imatge de la cara i l'aspecte durant el joc. D'aquesta forma un jugador pot escollir l'aspecte i el nom del seu avatar abans d'iniciar la partida. La finestra per crear l'avatar és la següent:

Un cop creat l'avatar (o els avatars en cas de tractar-se de mode dos jugadors o mode contra la màquina) s'ha d'escollir un mapa. Els mapes es carreguen al *listBox* obtenint-ne els noms des d'uns fitxers que conté els mapes que hi ha disponibles per cada mode de joc. En seleccionar algun dels mapes del *listBox* es mostren les seves dades al *groupBox* del costat.



Evidentment una partida no es pot iniciar si tots els jugadors que hi participen no tenen nom o avatar creat, o si no s'ha seleccionat un mapa. Un cop tot està a punt des d'aquest menú es crida el constructor de l'objecte mapa, que carrega les dades des dels fitxers de text per crear les diferents capes guardades en matrius, així com per crear els objectes Player, PC i Castle necessaris per començar una partida, i finalment passa aquest mapa a la finestra on es desenvolupa l'acció del joc i allà comença la partida.

Implementació – 19: Menú principal

Finalment, i ja per concloure amb el treball d'implementació d'aquest projecte, l'última cosa que necessita un joc: el menú principal. Aquest és simplement una nova finestra de Windows amb una imatge de fons, un títol i 3 botons: un per començar una partida, l'altre per veure els crèdit d'aquest projecte, i l'últim per sortir. El primer obre la finestra comentada en el punt anterior i a partir d'allà s'inicia una partida, mentre que el segon obre una altra finestra petita que mostra el següent:

La finestra del menú principal té l'aspecte que es mostra a la pàgina següent. Referent al títol del joc, no hi ha res especial a mencionar excepte que, a falta de millors idees, vaig prendre el títol d'un llibre que havia tingut fa un temps i que recentment he estat buscant i no el trobo. Simplement al tenir aquest títol al cap em va sonar un nom atractiu per un joc (ja que el nom és també un detall a tenir en compte per acabar de concedir al joc un aspecte homogeni i un caràcter determinat).

Implementació – Epíleg

Amb això doncs conclou l'apartat més extens de la memòria d'aquest Treball de Final de Carrera. L'evolució que aquest ha anat visquen des que era una figura que es desplaçava a velocitat sobrehumana per la pantalla fins a ser un avatar movent-se entre un mapa amb boscos i muntanyes, entrant a castells i combatent contra monstres ha set realment espectacular. Particularment aquest projecte m'ha aportat molt, no sols a nivell informàtic (on he adquirit un nombre de coneixements realment espectacular pel fet d'haver de treballar contínuament amb més objectes més complexos dels que estava acostumat a manipular), sino també a nivell personal, ja que ha set una oportunitat per probar de crear íntegrament un tipus de programa amb els que tota la vida he tingut relació i que mai havia intentat programar, a part d'experimentar amb quelcom ben diferent del que havia fet durant tota la carrera. Ara doncs, i ja per concloure, una breu visió a les possibles millores que podria tenir aquest projecte, junt amb la bibliografia consultada i un glossari amb tot un seguit de paraules que podrien presentar confusió.

7. Millores i conclusions

És evident que, tractant-se aquest projecte d'un joc, les millores que s'hi podrien fer són realment quantioses, així que s'intentarà restringir l'abast d'aquestes de tal forma que el joc segueixi mantenint la seva estructura i els seus aspectes fonamentals. Dit això, les millores proposades podrien ser:

- Aspecte gràfic del mapa: Veient com funciona el sistema de *tilesets*, es podria crear o aconseguir un seguit de *tilesets* que presentéssin un aspecte més elaborat, o fins i tot provinent de renderitzacions d'imatges en 3D. Això donaria al joc un aspecte gràfic molt més atractiu pels temps que corren en que la majoria de jocs són tridimensionals, o bidimensionals però amb uns gràfics molt cuidats (sense anar més lluny, un dels jocs més coneguts de l'empresa Blizzard anomenat Diablo està fet seguint un sistema de *tiles* com aquest, però utilitzant uns *sets* molt més elaborats).
- Aspecte gràfic dels combats: D'igual manera els combats podrien enriquir-se molt més gràficament utilitzant uns *battlers* que es poguéssin animar o convertint la perspectiva de les batalles en una perspectiva lateral de forma que permanentment es veiguéssin totes les criatures presents al camp de batalla.
- Sistema de joc: Les millores que refereixen al sistema de joc poden ser realment quantioses i poden variar segons les preferències de cada un. No tenen una base estríctament tècnica, i poden anar des d'augmentar el nombre d'atacs disponibles per cada criatura, a augmentar el nombre d'edificis que es poden construïr en un castell, incloure més tipus d'esdeveniments, etc.
- Intel·ligència artificial: La intel·ligència mostrada per l'oponent controlat per la màquina també es podria millorar permetent per exemple que aquest pogués enfrontar-se a oponents del mapa (realitzant prèviament la pertinent avaluació), o que no es mogués aleatòriament sino seguint uns principis determinats per una funció heurística.

- Guardar / cargar partida: Es podria implementar un sistema per guardar una partida durant el seu transcurs i recuperar-la posteriorment, guardant una còpia de l'estat de totes les matrius del mapa en aquell moment, així com de la informació dels jugadors, avatars i castells.
- Editor d'aspecte gràfic i sonor: Tal i com està estructurat el joc (en que pràcticament tota la part gràfica i sonora la pren parametritzada de fitxer) es podria crear un programa que fós com un assistent per crear (amb el format necessari) els arxius que parametritzen els gràfics, les criatures, els avatars, etc. i les músiques associades a cada situació.
- Mode multijugador on-line: Partint del mode multijugador implementat aquí es podria obrir un nou mode multijugador a través d'Internet (amb l'inconvenient del temps d'espera del torn contrari) en què dos jugadors s'enfrontéssin a través de la xarxa.
- Multijugador per més de dos jugadors: Tenint en compte que s'haurien de crear mapes per al nombre concret de jugadors (o adaptar els ja existents) es podria fer un multijugador apte per a més de dos jugadors, seguint el mateix sistema però augmentant el nombre de jugadors existents a la partida.

I com aquestes es podrien treure un gran nombre de millores que es podrien implementar a aquest projecte, doncs si pràcticament tot es pot millorar, quan es parla d'un joc el nivell de millora que s'hi pot afegir esdevé encara molt major. No obstant això i deixant de banda el que (aplicant tantes millores com hom desitgés) podria haver estat, un servidor està més que satisfet amb el resultat d'aquest projecte, tal i com ja s'ha dit anteriorment, tan informàtica com personalment. I ja per últim i per posar punt i final aquesta memòria, s'acabarà de la forma més adient pel tipus de porjecte que és:

Game Over.

Annex: Glossari

A

Avatar: S'anomena així a la representació a la pantalla del personatge que un jugador controla en un joc.

B

Battler: S'anomena battler a una imatge estàtica utilitzada per representar una criatura en el transcurs d'un combat.

C

Char: Abreviació de l'anglès *character*, en general no fa referència al caràcter com a personatge, sinó com la imatge que el representa a la pantalla.

Charset: Imatge gran que conté totes les imatges corresponents als diferents estats que pot tenir l'animació d'un o varis personatges, així com totes les seves posicions.

Criatura: En determinats jocs, s'entén com a criatura una entitat que representa un ens vivent i que d'alguna manera interactua amb el jugador, però essent la seva interacció restringida a àmbits molt concrets (sobretot combats).

D

DirectX: Conjunt de llibreries gràfiques i sonores distribuïdes per Microsoft que s'utilitzen en gran part en el món dels jocs, tant per implementar-los com per fer-los funcionar.

E

Estratègia, joc de: Vegeu *Joc d'estratègia*.

J

Joc d'estratègia: Malgrat la classificació dels jocs en gèneres pot ser molt ambígua, s'acostuma a anomenar així als jocs que presenten una certa necessitat de premeditar i planificar unes accions per part del jugador per tal d'assolir uns objectius.

Joc per torns: S'anomena així a un seguit de jocs que, en menor o major mesura, divideixen les accions que han de dur a terme els diferents jugadors en una sèrie de torns que se succeeixen un rere l'altre.

L

Llibreries: En informàtica s'entén per llibreries com un conjunt de recursos integrables en un llenguatge de programació o en un programa, i que proporcionen una sèrie de recursos determinats per a utilitzar en aquests.

M

Mapa: En un joc, regió finita de terreny per on es desplacen els avatars i on es du a terme l'acció d'aquest.

N

NPC: Sigles de *Non Playable Character*. S'anomena així a les entitats que participen en un joc però que no són controlables per cap jugador humà.

P

PC: Sigles de *Playable Character*. S'anomenen així les entitats que participen en un joc i que són controlades pels jugadors humans d'aquest.

Píxel: Unitat mínima que es pot pintar en una imatge o pantalla. A un píxel se li associa un únic color, i la unió contígua de varis píxels de diferents colors creen una imatge. El nombre de píxels que formen una imatge determinen el seu tamany.

Player: Jugador en anglès. S'entén com a jugador qualsevol persona que d'una manera o altra participi en un joc. Aquesta definició es podria extendre també en funció del context a un jugador controlat per la màquina.

S

SDK: Sigles de *Software Development Kit*. Es tracta d'un paquet que conté un conjunt de llibreries destinades a la implemenatació de software.

Sprite: S'anomena així una espècie de capa que té associada una determinada textura i que pot ésser pintada i mostrada a la pantalla.

T

Texture: S'anomena així qualsevol imatge que es carrega des d'un fitxer per tal de ser pintat llavors a través d'un *sprite*. Un *tile*, un *battler* o un *char* són exemples comuns de textures.

Tile: S'anomena així a una imatge, generalment de tamany molt petit, que permet dibuixar-se contíguament per representar una superfície molt major (d'aquí ve que la traducció de *tile* és rajola, per la similitud que guarda aquest sistema amb una estància coberta per rajoles).

Tileset: Imatge gran que conté tot un seguit de *tiles*.

Torn: En els jocs per torns, aquest és la unitat temporal durant la qual un jugador (humà o no) té disponible el control d'una o determinades entitats.
Torns, joc per: Vegeu *Joc per torns*.

Bibliografia

Pàgines web:

http://gregs-blog.com/2008/02/26/managed-directx-c-graphics-tutorial-1-getting-started/

http://www.toymaker.info/Games/html/2d_elements.html

http://www.gamedev.net/community/forums

http://www.garagegames.com

http://www.c-sharpcorner.com/

http://einfall.blogspot.com/

http://www.euclideanspace.com/software/games/twod/dx9cs/index.htm

http://geekswithblogs.net/jolson/archive/2004/02/16/2174.aspx

http://blogs.msdn.com/coding4fun/archive/2007/02/20/1727608.aspx

Llibres:

BUONO, Salvatore A. (2003): *C# and Game Programming: A Beginner's Guide*. Londres. AK Peters, Ltd.