DESARROLLO DE UN VIDEOJUEGO BASADO EN INTELIGENCIA ARTIFICIAL

Memoria del proyecto
de Ingeniería Técnica en Informática de Sistemas
realizado por
Juan Pedro Rodríguez Extremera
y dirigido por
Vicenç Soler Ruiz

Escuela de Ingeniería
Sabadell, Septiembre de 2011

El abajo firmante, **Vicenç Soler Ruiz**,
profesor de la Escuela de Ingeniería de la UAB,

CERTIFICA:
Que el trabajo al que corresponde la presente memoria
ha sido realizado bajo su dirección por
Juan Pedro Rodríguez Extremera
Y para que conste firma la presente.
Sabadell, Septiembre de 2011

Firmado: **Vicenç Soler Ruiz**

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

Títol del proyecto: Desarrollo de un videojuego basado en inteligencia artificial	
Autor[a]: Juan Pedro Rodríguez Extremera	**Data:** Septiembre 2011
Tutor[a]/s[es]: Vicenç Soler Ruiz	
Titulació: Ingeniería técnica en informática de sistemas	

Paraules clau (mínim 3)

- Català: Intel·ligència artificial, videojocs, Bennu
- Castellà: Inteligencia artificial, videojuegos, Bennu
- Anglès: Artificial Intelligence, videogames, Bennu

Resum del projecte (extensió màxima 100 paraules)

- Català: El projecte *Desarrollo de un videojuego basado en inteligencia artificial* té com objectiu el desenvolupament d'un videojoc d'estratègia on el jugador haurà de planificar una defensa capaç de resistir l'atac de l'enemic. El jugador haurà de triar com emprar els seus recursos per col·locar diferents tipus de terreny i diferents tipus de soldats. Posteriorment, quan aparegui l'enemic, els soldats actuaran d'acord als seus mecanismes de presa de decisions (encara que alguns soldats poden ser controlats pel jugador), igual que els enemics.

- Castellà: El proyecto *Desarrollo de un videojuego basado en inteligencia artificial* tiene como objetivo el desarrollo de un videojuego de estrategia donde el jugador deberá planificar una defensa capaz de resistir el ataque enemigo. El jugador deberá elegir como emplear sus recursos para colocar diferentes tipos de terreno y diferentes tipos de soldados. Posteriormente, cuando aparezca el enemigo, los soldados actuarán de acuerdo a sus mecanismos de toma de decisiones (aunque algunos soldados pueden ser controlados por el jugador), al igual que los enemigos.

- Anglès: The project *Desarrollo de un videojuego basado en inteligencia artificial* aims at the development of a strategy game where the player must plan a defense able to withstand the enemy attack. The player must choose how to use his resources to place different types of terrain and different types of soldiers. Later, when the enemy army arrives , the soldiers will act according to their decision-making mechanisms (though some soldiers may be controlled by the player), as well as enemies.

Índice

1 Introducción

El proyecto que se presenta a continuación consiste en el desarrollo de un videojuego de estrategia. El objetivo del juego es construir una defensa capaz de repeler el ataque del ejército enemigo.

El videojuego consta de tres partes principales: Menú principal, Modo de preparación y Modo de Batalla.

El Menú principal es el punto donde se inicia el juego. Desde el se podrá iniciar una nueva partida, cargar un mapa anteriormente guardado, ver los créditos o salir del programa. Si se inicia una nueva partida o se carga un mapa se pasara al Modo de preparación.

En el Modo de preparación se construirá el mapa. Aquí el jugador dispondrá de una cierta cantidad de puntos que podrá invertir en elementos de terreno o en soldados. Cada elemento o soldado tiene diferentes características. Para organizar una buena defensa el jugador debe saber como combinar mejor los diferentes tipos de terrenos y soldados.

En el Modo de preparación se podrá acceder a un menú desde el cual se podrá guardar el mapa para una posterior carga, volver al Menú principal o pasar al Modo de Batalla.

En el modo de Batalla el ejército enemigo aparecerá por oleadas por el portal de entrada. La defensa del jugador será puesta a prueba. Si el jugador lo desea puede tomar el control de uno de los soldados de tipo comandante para participar en la batalla. La batalla se pierde si el enemigo llega al final de la pantalla. Si el jugador consigue eliminar a todo el ejército enemigo habrá conseguido la victoria.

1.1 Estado del arte

En la actualidad la popularidad de los videojuegos no deja de crecer. Pueden encontrarse en una gran cantidad de formatos y plataformas. Desde simples juegos de redes sociales hasta grandes superproducciones con altos presupuestos y un gran número de trabajadores de todo tipo detrás, desde programadores hasta grafistas, guionistas, técnicos de sonido, etc...

El mundo del desarrollo de videojuegos es muy competitivo. Todos los grandes desarrolladores compiten por ofrecer el juego más original, el apartado gráfico más espectacular, la mejor banda sonora y, en conjunto, la mejor experiencia posible.

Ofrecer un juego “diferente” es cada vez más difícil. De hecho, muchos de los grandes proyectos que se desarrollan son secuelas de juegos o juegos muy parecidos a otros pero con mejoras en varios apartados.

Sin embargo, hoy existen varias tecnologías que hacen posible el desarrollo de videojuegos de calidad sin disponer de presupuestos desorbitados. Los videojuegos de este tipo no podrán competir con los productos de las grandes desarrolladoras en el apartado gráfico o sonoro, pero afortunadamente esos aspectos no son los más

importantes. Un videojuego de bajo presupuesto puede obtener un gran éxito siempre que sea innovador, diferente o, simplemente, muy divertido.

El desarrollo de un juego diferente a lo convencional es algo arriesgado, por lo que las grandes compañías prefieren desarrollar juegos más “standard”. Esto hace que el catalogo de videojuegos actual este repleto de gran cantidad de juegos con muchas similitudes entre si.

1.2 Inspiración

El objetivo del proyecto es el desarrollo de un juego diferente, algo innovador que no se límite a copiar otro juego de gran éxito.

Uno de los métodos más usuales a la hora de diseñar un nuevo tipo de juego es la mezcla de géneros. Esto puede dar lugar a grandes éxitos de ventas por su originalidad, pero también puede dar pie al fracaso, todo depende de como se lleve a cabo esa mezcla.

La mezcla de géneros es, pues, algo arriesgado. Aun así eso es lo que busca este proyecto exactamente.

La principal inspiración de este proyecto ha sido la famosa saga *Metal Slug*. El modo de juego en Metal Slug se basa en que el personaje que controla el jugador porta armas extremadamente poderosas mientras se enfrenta a una gran cantidad de

enemigos. Además, como en la mayoría de estos juegos, con sólo el contacto de un arma enemiga, el personaje muere y pierde las armas que porta. El jugador dependerá de su habilidad para derribar a un gran número de tropas enemigas.

La idea es dar una vuelta de tornas al juego, convirtiendo al jugador en el malvado que, valiéndose de su ejército, intentara resistir la embestida del héroe, dando así un toque de estrategia a este genero.

1.3 Motivaciones personales

El principal motivo por el cual decidí hacer este tipo de proyecto es vocacional. Siempre he sido un aficionado a los videojuegos y , como programador, creo que es un paso lógico pasar del simple uso del videojuego a su desarrollo.

Desde el principio vi una gran oportunidad en el proyecto final de carrera, ya que mi objetivo a largo plazo es entrar en la industria del videojuego. Una primera toma de

contacto con un proyecto serio de desarrollo de un videojuego me dará una valiosa experiencia.

Realizar este proyecto me ayudará a aprender nuevos lenguajes y tecnologías útiles para el desarrollo de videojuegos y, por tanto, mejorar como programador.

Además, mi intención dista mucho de abandonar el proyecto una vez presentado. Al contrario. Tras presentar el proyecto seguiré trabajando en él, para mejorarlo. Será el primero de muchos proyectos que, más adelante, me ayudarán a crecer como programador y demostrarán mis habilidades como desarrollador de cara a entrar en la industria.

1.4 Estructura de la memoria

Este documento esta estructurado en ocho capítulos (incluyendo este mismo). A continuación se expone una breve descripción de cada uno de ellos:

– **2 Estudio de viabilidad:** Aquí se expondrá el estudio de viabilidad hecho previo al comienzo de este proyecto. En él se expone cuales eran los requisitos a cumplir por el proyecto, la planificación inicial, los posibles riesgos, planes de contingencia, etc..

– **3 Herramientas utilizadas:** En este capitulo se proporciona información detallada sobre las principales herramientas utilizadas en el desarrollo de este proyecto.

– **4 Análisis de la aplicación:** Aquí se da información detallada del funcionamiento interno de la aplicación desarrollada en este proyecto.

- **5 Instrucciones de uso:** En este capitulo se exponen las instrucciones de uso del videojuego desarrollado en este proyecto.

- **6 Problemas y soluciones:** Aquí se se describirán con detalle cuales han sido los problemas surgidos durante el desarrollo del proyecto y cuales han sido las soluciones aplicadas.

- **7 Conclusiones:** En este capitulo se exponen las conclusiones del proyecto.

- **8 Bibliografía:** Se exponen cuales han sido las principales fuentes de información utilizadas durante el desarrollo del proyecto.

2 Estudio de viabilidad

2.1 Introducción

En este estudio de viabilidad veremos una descripción general del proyecto así como de los objetivos del mismo y los requisitos para llevarlo a cabo.

Tipología: Proyecto de desarrollo de sistemas de software.

Palabras clave: videojuego, BennuGD, inteligencia artificial.

Descripción: Se quiere desarrollar un videojuego en el cual el jugador ha de planificar y llevar a cabo la defensa de una base contra un ejército enemigo controlado por la CPU. En este juego, el usuario hará el papel del "malo" y tendrá de evitar que el héroe llegue hasta el final de la pantalla preparando trampas, situando tropas, dirigiendo la defensa y, si todo falla, luchando contra el héroe y sus artefactos. El héroe tendrá a su alcance una serie de artefactos para volar, teletransportarse, armas etc.. pero el usuario no sabrá cuales son hasta que el héroe los utilice.

Objetivos:

Los objetivos del proyecto son los siguientes:

1- La interfaz de usuario ha de ser amigable, fácil de utilizar.
2- Tiene que haber una buena variedad de opciones disponibles, tanto para el jugador como para el enemigo.
3- El programa debe ser eficiente, no contener errores, no usar demasiados recursos...
4- El juego debe ser ampliable para poder añadir nuevos tipos de terrenos, tropas, gráficos, etc...

Referencias:

Normativa de proyectos de ingeniería técnica:

Http://www.uab.cat/Document/330/254/projectes-normativa20090630.pdf

Producto y documentación del proyecto:

Se librará una aplicación informática con la licencia correspondiente.

Se elaborará una memoria del proyecto.

2.2 Requisitos del proyecto

Los requisitos del proyecto son los siguientes:

Requisitos funcionales

1. El usuario debe poder colocar un conjunto de elementos de terreno y soldados en el nivel, además de poder gestionarlos más adelante.
2. El usuario ha de ser capaz de guardar sus niveles creados para poder cargarlos más adelante para así modificarlos y reutilizarlos.
3. El usuario debe poder pausar/salir del juego en todo momento.

Requisitos no funcionales

1. Cumplimiento de la LOPD en aquellos aspectos que hacen referencia a los ficheros de datos y a los derechos de los clientes.
2. Los recursos utilizados por la aplicación han de estar ajustados a la medida de la entidad.
3. Usabilidad: la interfaz ha de cumplir la *ISO 9241: Ergonòmic requeriments for office work with visual display terminals.*

Restricciones del sistema

1. La aplicación ha de adaptarse al sistema físico disponible en la entidad.
2. El proyecto ha de estar finalizado antes del 20 de Junio de 2011.

Priorización de los requisitos funcionales

	RF1	RF2	RF3
Critico	X		
Primario		X	
Secundario			X

Priorización del requisitos no funcionales

	RNF1	RFN2	RFN3
Critico	X		
Primario		X	X
Secundario			

2.3 Alternativas y selección de la solución

A la hora de escoger que herramientas se utilizaran en el desarrollo del proyecto se ha de tener en cuenta tanto la capacidad de la herramientas para cumplir los objetivos deseados como la facilidad a la hora de utilizarlas en el desarrollo del proyecto.

La primera opción fue utilizar java con librerías gráficas, ja que ofrece un entorno conocido y portabilidad al desarrollo del proyecto.

Pero aun estando familiarizado con el entorno java, este no es el caso con las librerías gráficas y de sonido. Además, el punto crucial del proyecto, el desarrollo de la inteligencia artificial ocupará una parte importante del tiempo, por lo que es importante intentar ahorrar tiempo en otros aspectos del proyecto.

Con tal de ahorrar tiempo en otras tareas, se buscó herramientas que ofrecieran facilidades en tales aspectos. El lenguaje de programación Bennu Game Development (a partir de ahora referenciado como BennuGD o simplemente Bennu) , ofrece un entorno dirigido al desarrollo de videojuegos en dos dimensiones, de manera que todos los aspectos gráficos y sonoros del desarrollo del videojuego pueden ser tratados de forma más sencilla, dejando más tiempo para las tareas más importantes. Además también aporta una gran portabilidad.

Es por eso que se ha escogido BennuGD como herramienta principal del desarrollo de este proyecto.

Para gran parte del material gráfico y sonoro del juego se recurrirá a la página opengameart.org. En dicha web se puede encontrar una gran cantidad de material bajo licencias libres, lo cual ahorrará mucho tiempo y esfuerzo. Para poder utilizar dicho material la licencia del juego tendrá que ser libre al mismo tiempo, y se tendrá

que acreditar las fuentes de todo el material usado. Ninguna de esas dos condiciones presenta un obstáculo para el desarrollo del proyecto.

El tratado de las imágenes se realizará con el editor de imágenes Gimp por su gran versatilidad y porque ofrece un entorno conocido.

Para asegurar la seguridad de la información del mismo proyecto se usará Dropbox, un disco duro virtual que ofrece backup de la información y la posibilidad de trabajar desde diferentes terminales.

Como puede observarse todas las herramientas necesarias son gratuitas.

2.4 Evaluación de riesgos

Aquí se listarán los riesgos que se podrían encontrar durante la elaboración del proyecto, en qué fase de este podrían aparecer y cómo afectarán al proyecto. También veremos el plan de contingencia para cada riesgo.

Listado de posibles riesgos

R1. Planificación temporal optimista: estudio de viabilidad. No se acaba en la fecha prevista, aumentan los recursos necesarios para la realización del proyecto.

R2. Ausencia de alguna tarea necesaria: estudio de viabilidad. No se cumplen los objetivos del proyecto.

R3. Cambio de requisitos: estudio de viabilidad, análisis. Retraso en desarrollo y cambio en los resultados obtenidos.

R4. Herramientas de desarrollo inadecuadas: implementación. Atraso en la finalización del proyecto, se obtiene un resultado de una calidad menor a la esperada, ...

R5. Ausencia de implementación de medidas de seguridad: estudio de viabilidad, análisis, desarrollo. Pérdida de información, incumplimiento legal.

R6. Abandono del proyecto antes de su finalización: en cualquier fase. Frustración.

Plan de contingencia:

R1. Hacer la entrega del proyecto en la próxima convocatoria o, si hace falta, el próximo curso.

R2. En caso de no poder implementar con éxito la inteligencia artificial del héroe, se podría modificar el juego para que fuera un segundo jugador quién controle el héroe, o hacer una modificación totalmente diferente a la estructura del juego.

R3. Hacer la entrega del proyecto en la próxima convocatoria o, si hace falta, el próximo curso.

R4. Búsqueda de herramientas adecuadas. En caso de que esto atrase demasiado el desarrollo del proyecto se haría la entrega del mismo en la siguiente convocatoria. En caso de que no se encontrara ninguna herramienta adecuada se tendría que replantear el proyecto, hasta el punto de poder llegar a abandonarlo.

R5. Para evitar la pérdida de información se harán copias de seguridad del proyecto forma regular

R6. No tiene solución.

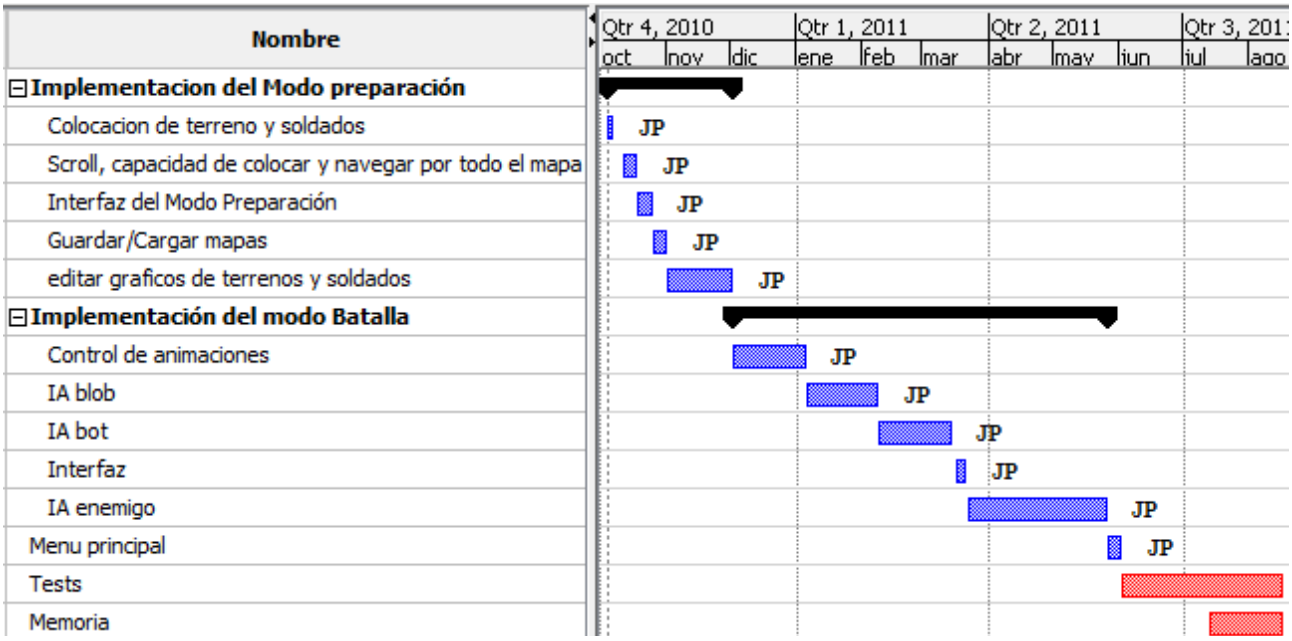
2.5 Planificación

La siguiente tabla resume cuanto tiempo se prevé que ocupe cada una de las tareas que compone el desarrollo del proyecto.

Desarrollo del proyecto	207 días
– Implementación del Modo Preparación	39 días
– – Colocación de terreno y soldados	4 días
– – Scroll	5 días
– – Interfaz del Modo Preparación	5 días
– – Guardar/Cargar mapas	5 días
– – Editar gráficos de terrenos y soldados	20 días
– Implementación del Modo Batalla	103 días
– – Animaciones	20 días
– – IA blob	20 días
– – IA bot	20 días
– – Interfaz	3 días
– – IA enemigo	40 días
– Menú principal	5 días
– Testing	40 días
– Memoria	20 días

Estos valores vienen dados por una planificación de 1 hora diaria, teniendo en cuenta los fines de semana. Se ha hecho así teniendo en cuenta el tiempo que se empleara en otras asignaturas y deberes no relacionados con el proyecto. Es, por lo tanto, una planificación realista (e incluso pesimista) que da un amplio margen al compararla con la realidad, pues es muy posible que se le pueda dedicar más tiempo al proyecto del marcado en la planificación, sobretodo en verano.

El diagrama de Gantt obtenido por esta planificación es el siguiente:



2.6 Conclusiones

Tal y como se ha visto, los objetivos son asequibles, existen planes de contingencia para todo posible problema que pudiera surgir durante el desarrollo del proyecto, se dispone de las herramientas adecuadas y la planificación ofrece margen de maniobra en caso de imprevistos. Por todo ello se puede concluir que el proyecto es viable.

2.7 Desarrollo del proyecto

Durante el desarrollo del proyecto la estructura del videojuego cambió por varios motivos. El primero de ellos, la dificultad de la implementación de la IA del héroe no solo se planteaba como muy complicada sino que requeriría una gran cantidad de

tiempo. El segundo motivo fue que tras pensarlo deliberadamente, era posible que al jugador no le gustara ver como gran parte de su defensa caía bajo un único personaje. Había riesgo de que el juego se volviera frustrante para el jugador.

Para solucionarlo se substituyó el héroe enemigo por un ejército, un conjunto de IA más simples, que requerirían menor tiempo y dificultad de implementación. Por parte del jugador, no es lo mismo ver como un personaje arrasa con tu defensa que ver, en cambio, como un ejército lucha contra el tuyo, con bajas en ambas partes, con una mayor sensación de igualdad entren ambos bandos.

Este cambio supuso un retraso en el la planificación del proyecto y un cambio de requisitos y objetivos.

Aparte de eso el desarrollo del proyecto ha seguido la planificación sin más incidencias.

3 Herramientas utilizadas

3.1 BennuGD

BennuGD es un sencillo lenguaje de programación libre y multiplataforma (existen versiones oficiales para sistemas Windows, GNU/Linux y OpenWiz) de tipo compilado/interpretado y que está específicamente diseñado para crear y ejecutar cualquier tipo de juego 2D. Tal como dice su página web oficial: Http://www.bennugd.org , Bennu es un entorno de desarrollo de videojuegos de "alto nivel" que da especial énfasis a la portabilidad del código, (proporcionando al desarrollador la posibilidad de escribir su programa una sola vez pero pudiéndolo ejecutar en múltiples plataformas sin modificar absolutamente nada), y también a la modularidad, (existiendo múltiples librerías que extienden y complementan la funcionalidad básica de Bennu: librerías de red, librerías 3D, de renderizado de textos, de acceso a bases de datos y archivos XML, etc). Se puede decir, pues, que con Bennu tanto los programadores noveles como los avanzados encontrarán una inestimable herramienta para sus lograr hacer realidad sus creaciones.

El lenguaje está claramente orientado al manejo de gráficos 2D; es decir, que el propio lenguaje se preocupa de dibujar los gráficos en pantalla, ahorrándole al programador el trabajo que eso supone, que es mucho. Bennu incluye de serie un motor de renderizado 2D por software que convierte la programación de juegos en una tarea fácil pero potente. En general, cualquier juego que no emplee objetos 3D real es posible realizarlo con Bennu sin necesidad de ninguna librería extra.

El lenguaje en sí soporta muchas de las características comúnmente encontradas en los lenguajes procedimentales, tales como múltiples tipos de dato , punteros, tablas multidimensionales, estructuras, y las sentencias habituales de control de flujo. De hecho, Bennu se puede interpretar como un recubrimiento, una capa superior de la librería SDL, que encapsula, que oculta la dificultad intrínseca que tiene a la hora de programar con ella en C.

3.1.1 Características

*Dibujo rápido de sprites con rotado, escalado, animación y grados de transparencia.
*Detección de colisiones a nivel de píxel

*Procesos (programación multihilo)

*Entrada por teclado, ratón y joystick

*Modos de color de 8, 16 y 24/32 bits

*Soporte del formato gráfico Png, entre otros.

*Soporte de los formatos de sonido Wav sin compresión y Ogg Vorbis.

*Rutinas de scroll tipo "parallax"

*Múltiples regiones en pantalla con o sin scroll

*Recorrido de caminos inteligente

*Rutinas matemáticas y de ordenación

*Rutinas de acceso y manipulación de directorios y ficheros

*Consola de depuración

*Uso de temporizadores

*Comprobación de expresiones regulares y manejo de cadenas de caracteres complejas

*Manejo dinámico de memoria

*Soporte para Modo 7

*Soporte para el cifrado de recursos mediante criptografía simétrica

*Soporte de librerías externas (en Windows, *.dll; en GNU/Linux, *.so).

*Multiplataforma: funciona de forma oficial en todos los Windows (incluido Windows 7), en GNU/Linux sobre chips Intel y en la consola GP2X Wiz (Http://www.gp2xwiz.com). Pero es factible portar Bennu a múltiples plataformas más, como por ejemplo MacOSX, PSP, Xbox, Wii y PlayStation, gracias a que Bennu está basado a más bajo nivel en la librería gráfica SDL (ver Apéndice A) , la cual es portable a todos estas plataformas mencionadas. De hecho, existe un port no oficial de Bennu para la consola Wii, disponible en Http://code.google.com/p/bennugd-wii.

3.1.2 Historia

En la década de 1990, Daniel Navarro Medrano creó una herramienta orientada a la creación de videojuegos de 32 bits bajo MS-DOS. El nuevo lenguaje, de nombre DIV Games Studio, combinaba características de C y Pascal con un entorno completo que permitía la creación y edición de todos los aspectos de los proyectos: programación, edición gráfica y sonora y un largo etc.

Fénix, inicialmente bajo el nombre DIVC y de naturaleza GNU, apareció de la mano de José Luis Cebrián como una herramienta capaz de compilar y ejecutar esos juegos en Linux. El nombre fue cambiado en la versión 0.6 del compilador, que además introducía otras mejoras, como la aparición de un fichero intermedio entre el entorno de compilación y el entorno de ejecución. Ya no era necesario distribuir el código fuente de un juego para poder jugar a los juegos. La ventaja principal de esa práctica

(similar en concepto a Java) era clara, compilar en una plataforma y ejecutar en muchas.

En la versión 0.71 el proyecto quedó parado, lo que dio lugar a múltiples versiones derivadas que corregían fallos o añadían nuevas características.

La versión oficial de Fénix fue retomada por Slàinte en el año 2002, viejo conocido de la comunidad DIV por ser el webmaster de una de las páginas web más importantes para la comunidad, quien continuó el proyecto bajo el nombre de Fénix - Proyecto 1.0 al que pronto se reincorporaría su creador y cuyo primer objetivo era limpiar el compilador de errores y estabilizarlo. Desde entonces el compilador ha sufrido numerosos cambios y mejoras, dejando de un lado la compatibilidad con el lenguaje DIV.

Más tarde, tras un largo tiempo sin modificaciones, en el año 2006, Fénix fue retomado por el hacker argentino SplinterGU, el mismo que implementó el primer sistema de dlls. Tras muchas idas y vueltas, SplinterGU decidió crear un fork de Fénix, con grandes cambios internos como la adopción de un sistema modular, pero que mantendría la compatibilidad con su predecesor. Así nace Bennu, que luego añadiría a su nombre "GD" (Game Development) debido a que ya existía otro proyecto con el nombre original.

3.2 GIMP

GIMP (GNU Image Manipulation Program) es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de GNU.

Es el programa de manipulación de gráficos disponible en más sistemas operativos (Unix, GNU/Linux, FreeBSD, Solaris, Microsoft Windows y Mac OS X, entre otros).

3.2.1 Características

GIMP es un programa de manipulación de imágenes que ha ido evolucionando a lo largo del tiempo, ha ido soportando nuevos formatos, sus herramientas son más potentes, además funciona con extensiones o plugins y scripts.

GIMP usa GTK+ como biblioteca de controles gráficos. En realidad, GTK+ era simplemente al principio una parte de GIMP, originada al reemplazar la biblioteca comercial Motif usada inicialmente en las primeras versiones de GIMP. GIMP y GTK+ fueron originalmente diseñados para el sistema gráfico X Window ejecutado sobre sistemas operativos tipo Unix. GTK+ ha sido portado posteriormente a Microsoft Windows, OS/2, Mac OS X y SkyOS.

GIMP permite el tratado de imágenes en capas, para poder modificar cada objeto de la imagen en forma totalmente independiente a las demás capas en la imagen, también pueden subirse o bajarse de nivel las capas para facilitar el trabajo en la imagen, la imagen final puede guardarse en el formato xcf de GIMP que soporta capas, o en un formato plano sin capas, que puede ser png, bmp, gif, jpg, etc.

Con GIMP es posible producir imágenes de manera totalmente no interactiva (por ejemplo, generar al vuelo imágenes para una página web usando guiones CGI) y realizar un procesamiento por lotes que cambien el color o conviertan imágenes. Para tareas automatizables más simples, probablemente sea más rápido utilizar un paquete como ImageMagick.

El nombre de GIMP en español se forma con las iniciales de Programa de Manipulación de Imágenes de GNU leídas de atrás para adelante.

Formatos soportados

GIMP lee y escribe la mayoría de los formatos de ficheros gráficos, entre ellos; JPG, GIF, PNG, PCX, TIFF, los de Photoshop, además de poseer su propio formato de almacenamiento de ficheros, XCF. Es capaz de importar ficheros en pdf. También importa imágenes vectoriales en formato SVG creadas, por ejemplo, con Inkscape.

3.3 Notepad++

Notepad++ es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación. Solo funciona en Microsoft Windows.

Se parece al Bloc de notas en cuanto al hecho de que puede editar texto sin formato y de forma simple. No obstante, incluye opciones más avanzadas que pueden ser útiles para usuarios avanzados como desarrolladores y programadores.

Se distribuye bajo los términos de la Licencia Pública General de GNU.

3.3.1 Características

- Coloreado y envoltura de sintaxis: si se escribe en un lenguaje de programación o marcado, Notepad++ es capaz de resaltar las expresiones propias de la sintaxis de ese lenguaje para facilitar su lectura.
- Pestañas: al igual que en muchos navegadores, se pueden abrir varios documentos y organizarlos en pestañas.
- Resaltado de paréntesis e indentación: cuando el usuario coloca el cursor en un paréntesis, Notepad++ resalta éste y el paréntesis correspondiente de cierre o apertura. También funciona con corchetes y llaves3
- Grabación y reproducción de macros.
- Soporte de extensiones: incluye algunas por defecto.

3.4 Dropbox

Dropbox es un servicio de alojamiento de archivos multiplataforma en la nube, operado por la compañía Dropbox. El servicio permite a los usuarios almacenar y sincronizar archivos en línea y entre computadoras y compartir archivos y carpetas con otros. Existen versiones gratuitas y de pago, cada una de las cuales con opciones variadas.

El cliente de Dropbox permite a los usuarios dejar cualquier archivo en una carpeta designada. Ese archivo es sincronizado en la nube y en todas las demás computadoras del cliente de Dropbox. Los archivos en la carpeta de Dropbox pueden entonces ser compartidos con otros usuarios de Dropbox o ser accedidos desde la página Web de Dropbox. Asimismo, los usuarios pueden grabar archivos manualmente por medio de un navegador web.

Si bien Dropbox funciona como un servicio de almacenamiento, se enfoca en sincronizar y compartir archivos. Tiene soporte para historial de revisiones, de forma que los archivos borrados de la carpeta de Dropbox pueden ser recuperados desde cualquiera de las computadoras sincronizadas. También existe la funcionalidad de conocer la historia de un archivo en el que se esté trabajando, permitiendo que una persona pueda editar y cargar los archivos sin peligro de que se puedan perder las versiones previas. El historial de los archivos está limitado a un período de 30 días, aunque existe una versión de pago que ofrece el historial ilimitado. El historial utiliza la tecnología de delta encoding. Para conservar ancho de banda y tiempo, si un

archivo en una carpeta Dropbox de un usuario es cambiado, Dropbox sólo carga las partes del archivo que son cambiadas cuando se sincroniza. Si bien el cliente de escritorio no tiene restricciones para el tamaño de los archivos, los archivos cargados por medio de la página Web están limitados a un máximo de 300 MB cada uno. Dropbox utiliza el sistema de almacenamiento S3 de Amazon para guardar los archivos y SoftLayer Technologies para su infraestructura de apoyo.

4 Análisis de la aplicación

4.1 BennuGD

Para entender el funcionamiento de la aplicación antes hay que comprender como funciona la principal herramienta usada para su desarrollo, es decir, el lenguaje BennuGD.

BennuGD puede describirse como un lenguaje procedimental, ya que se basa en su mayor parte en la creación y gestión de procesos para una ejecución multihilo.

Cada soldado, cada pieza de terreno, disparo o explosión es un proceso. De hecho casi toda imagen mostrada en el juego esta asignada a un proceso, desde los botones de la interfaz hasta el puntero del ratón mismo.

Esto es muy importante a la hora de entender que cada soldado es un proceso diferente que esta calculando su próxima acción (IA) de forma concurrente a otros procesos soldado.

BennuGD incorpora toda una serie de módulos con funciones muy útiles a la hora de desarrollar. Gracias a las funciones incluidas en estos módulos se facilita mucho, por ejemplo, la detección de colisiones entre procesos, el manejo de gráficos o el control de todo el apartado sonoro.

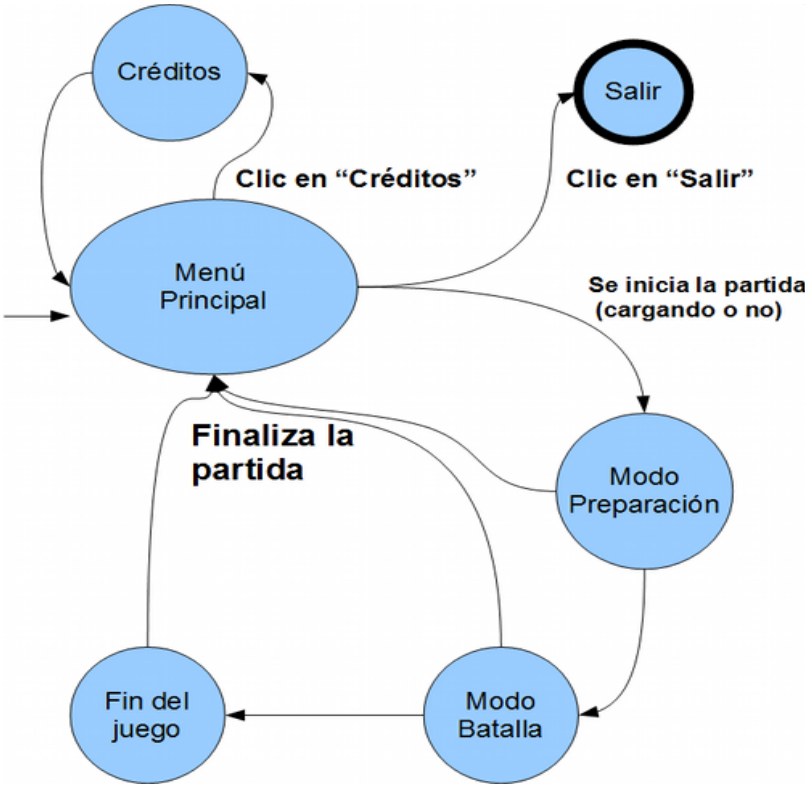
4.2 Funcionamiento interno del programa

En esta sección explicaremos el funcionamiento interno del programa paso a paso.

4.2.1 El proceso principal Main()

La clave del funcionamiento interno del programa es el proceso principal, Main(). Es este proceso el que controla en que estado del juego se encuentra la ejecución y coordina los diferentes procesos del mismo.

Este proceso es , a su vez, una maquina de estados finitos con 6 posibles estados:



A continuación se explicara con detalle cada estado del proceso principal, que procesos se ven involucrados y que hacen exactamente estos procesos.

4.2.1.1 Menú principal

Desde el estado inicial “Menú Principal”, se puede iniciar la partida entrando en el “Modo Preparación”, se pueden ver los créditos o se puede salir del programa.

Al iniciar este estado se reiniciarán todas las variables globales, se inicia la música del tema principal y se establece el fondo de pantalla. Como curiosidad para conseguir el efecto del fondo de pantalla se asigno al proceso Main() una imagen algo más grande que la pantalla, después, utilizando una variable local del proceso Main() que controla el ángulo del gráfico, se hizo rotar dicha imagen.

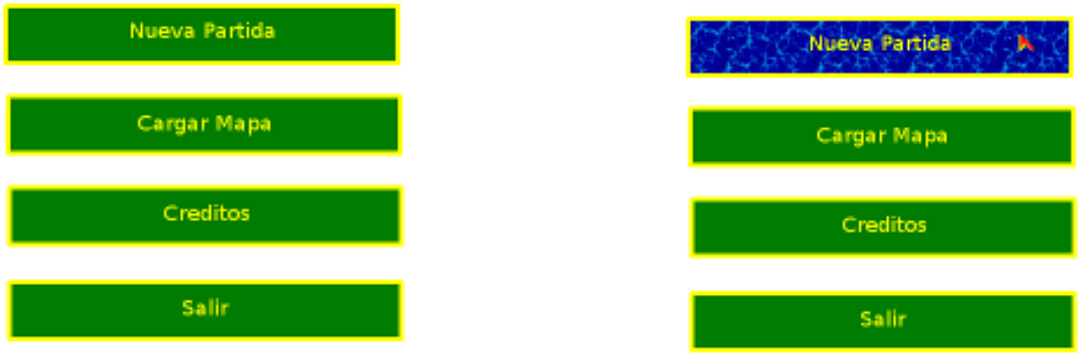
En este estado del programa intervienen los siguientes procesos:

- **Titulo():** Muestra el titulo del juego.

- **Puntero():** Proceso que sigue el puntero del mouse mostrando una imagen. Resulta útil tener un proceso de este tipo para facilitar la interacción entre diferentes procesos y el mouse.

- **Menú():** Este proceso controla los menús del juego en todos los estados del proceso principal. Para ello muestra un gráfico con el menú y, dependiendo de en que zona se encuentre el mouse cambia ese gráfico (por ejemplo cuando el puntero pasa

por encima de un botón). Cuando se acciona un botón este proceso avisa al proceso principal para que este cambie de estado.



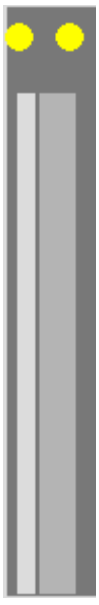
4.2.1.2 Modo Preparación

Al entrar en este estado se eliminan los procesos Titulo(), Menú() (el cual se creará cada vez que pulsemos la barra espaciadora de aquí en adelante), se borra el gráfico asignado al proceso Main() y se cambia la música, cargando la canción asignada a este estado. Después se inicia el scroll, colocando el fondo de pantalla, y se crean los procesos siguientes:

– **límite():** Establece el límite del mapa. Simplemente se limita el movimiento del scroll de forma que solo será posible avanzar o retroceder en el mapa cuando las dos instancias de este proceso (una situada al principio y otra al final del mapa) estén fuera de pantalla. Estos procesos no pueden ser vistos por el usuario.

– **Entrada():** Se coloca al inicio de la pantalla el portal por el cual más adelante aparece el ejército enemigo. En este estado no hace nada en concreto, aparte de mostrar su gráfico.

– **Final():** Es la puerta de salida, el punto que debe alcanzar el ejército enemigo para obtener la victoria. En este estado no hace nada en concreto, aparte de mostrar su gráfico.

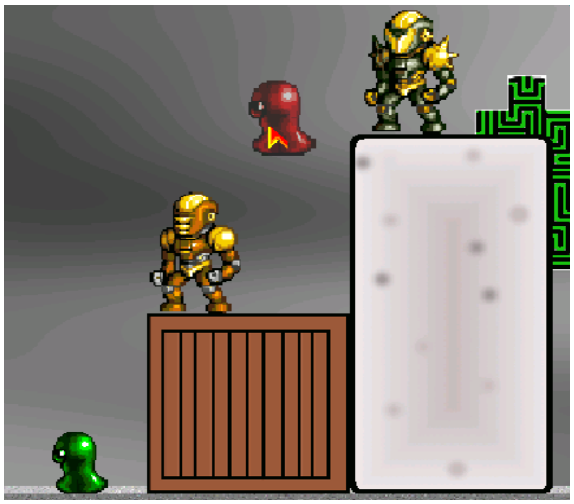


– **Casilla_insertar():** Hay cuatro procesos de este tipo (arriba a la izquierda de la pantalla). Al hacer clic sobre una de estas casillas un proceso del tipo elemento es creado, es así como se colocan las unidades y los elementos de terreno, como se verá

más adelante. Su gráfico muestra que se va a construir y cual es su coste en puntos de construcción.

– **Cambio_insertar():** Este proceso manda una señal a los procesos del tipo Casilla_insertar() para que cambien el tipo de elemento que se insertara si se hace clic en ellos. En concreto permite cambiar entre opciones de soldados y opciones de terreno.

– **Elemento():** Estos procesos hacen todos los cálculos necesarios para poder colocar de forma correcta todos los soldados y elementos de terreno que el usuario quiera incluir en su defensa. Dichos cálculos incluyen la detección de colisiones con otros elementos, la gravedad, el guardado del elemento en cuestión y, por supuesto, su eliminación por petición del jugador (o por colisión con el portal de entrada).



– **Menú():** En este estado del programa, el proceso menú se comporta igual que en el modo Menú principal, pero sus opciones cambian.

La opción “Listo” permite avanzar hacia el siguiente estado del programa, el Modo Batalla.

La opción “Guardar” permite, como el mismo nombre indica, guardar el mapa en su estado actual. Para ello se modifica una variable global llamada “orden_guardar”. Cuando los procesos del tipo elemento detectan que esta variable ha sido cambiada guardan sus respectivos datos en una estructura de datos creada para tal fin. Cuando el ultimo de los procesos ha guardado sus datos en dicha estructura, la estructura misma es guardada en un fichero de extensión *.dat en el interior de la carpeta “saves” junto a otros dos archivos que guardan otro tipo de información igual de importante a la hora de recuperar el mapa guardado.

Haciendo clic en “Continuar” saldremos del menú para seguir con la preparación de la defensa.

Finalmente con la opción “Salir” volveremos al estado inicial del programa, “Menú principal”.

Además de lo anteriormente mencionado, en este estado se muestra un texto en la parte superior derecha que indica la cantidad de puntos de construcción de los que dispone el jugador.

4.2.1.3 Modo Batalla

Al entrar en este estado, la música vuelve a cambiar, se eliminan todos los procesos relacionados con la inclusión de elementos en el escenario (casilla_insertar() y cambio_insertar()), se elimina el texto que muestra la cantidad de puntos de construcción y, lo que es más importante, todos los procesos del tipo elemento() son eliminados.

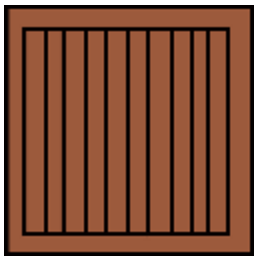
Antes de ser eliminados, los procesos del tipo elemento() crean un proceso del tipo soldado o terreno. El tipo de proceso que crearán dependerá de su gráfico. Un elemento con el gráfico de un bot, por ejemplo, creará un proceso del tipo bot() en su misma posición.

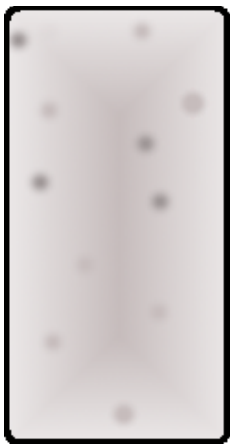
Una vez hecho esto, estos serán los procesos que intervendrán en este estado del programa:

– **Blob(), s_Blob(), Bot(), s_Bot():** Son los proceso que controlan los soldados del jugador. El comportamiento de estos proceso esta determinado por la inteligencia artificial que se explica más adelante. Los procesos del tipo s_Blob() y s_Bot() corresponden a los soldados comandantes. La diferencia entre estos y los soldados normales radica en su poder (mayor fuerza, vida, potencia de disparo...) y en la capacidad de poder ser controlados por el jugador en mitad de la batalla.

– **Flecha() y Cam():** Estos dos procesos intervienen a la hora de controlar a un comandante. El proceso Flecha() señala que soldados esta siendo controlado. Mientras, el proceso Cam() controla la cámara, asegurándose que nunca se sobrepasa el límite del mapa. El proceso Flecha() tiene asignado el siguiente gráfico (flecha amarilla sobre el soldado), mientras que el proceso Cam() es invisible al usuario.

– **Caja() y Muro():** estos procesos se comportan como lo harían una caja o un muro de piedra con una física muy simple. Caen con la gravedad y si por fortuna o desgracia hay algún proceso debajo, este será aplastado.

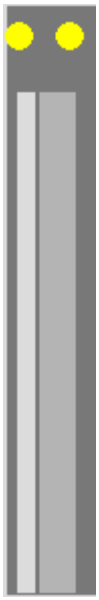




– **Possum(), s_Possum(), Mage(), s_Mage():** Son los proceso que controlan los soldados enemigos. El comportamiento de estos proceso esta determinado por la inteligencia artificial que se explica más adelante. Los procesos del tipo s_Possum() y s_Mage() corresponden a los soldados enemigos comandantes. La diferencia entre estos y los soldados normales radica en su poder (mayor fuerza, vida, potencia de disparo...).

– **Entrada():** En este estado la entrada es, de hecho, quien crea los procesos enemigos. Este proceso dispone de una cantidad de puntos de construcción iguales a los que disponía el jugador y los reparte de forma aleatoria entre los diferentes tipos de soldado enemigo. Este reparto aleatorio sigue una pauta de probabilidades, de modo que hay un 60% de que el proceso creado se un possum, un 20% de que sea un mago, un 10% de que sea un comandante possum y un 10% de que sea un comandante mago.

– **Final():** En este estado ,si este proceso detecta una colisión con un proceso soldado enemigo, el proceso principal pasara al siguiente estado “Game Over”.



– **Menú():** En este estado del programa, el proceso menú se comporta igual que en el Modo Preparación, pero sus opciones cambian.

Las dos opciones posibles permiten hacer lo mismo que en Modo preparación, es decir, continuar con la batalla o salir al menú principal.

Además de lo anteriormente mencionado, en este estado se muestra un texto en la parte superior derecha que indica la puntuación que ha obtenido el jugador hasta el momento.

4.2.1.4 Game over

Para bien o para mal la partida ha acabado. Todos los procesos del tipo soldado (amigo o enemigo) que sigan vivos serán eliminados, al igual que los límites, la entrada y el final. Se para el scroll y se limpia la pantalla de cualquier imagen, se para la música y cualquier otro efecto de sonido... En definitiva, el único proceso que se mantiene en funcionamiento es el principal.

Aquí se nos mostrará una serie de textos indicándonos el resultado de la batalla, la puntuación obtenida y como volver al Menú Principal (pulsando las teclas “esc” o enter”).

4.2.1.5 Créditos

Aquí se podrá ver quien ha intervenido en el desarrollo del proyecto de forma directa o indirecta. Se elimina el proceso Menú() y se cambia el gráfico asignado al proceso principal para mostrar los créditos.

4.2.1.6 Salir

Se eliminan todos los procesos y se liberan todos los recursos descargando de memoria los archivos fpg, los sonidos y la música.

4.2.2 IA (Inteligencia Artificial)

Una de las claves de este proyecto reside en el desarrollo de la IA. El primer paso a dar fue elegir una serie de prioridades que la IA debía cumplir.

- Dado que el número de soldados simultáneos en una partida será grande, la IA de cada uno de estos soldados ha de tener un **coste mínimo** para el sistema.

- Aunque el comportamiento de los soldados vaya a ser simple, no debe de caer en la estupidez, es decir, las **acciones de los diferentes soldados han de ser lógicas**.

Una vez determinado esto, faltaba decidir que método se usaría para la construcción de las diferentes inteligencias.

La primera opción, y la que se acabo adoptando, fue la de usar una maquina de estados finitos.

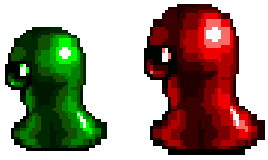
Una máquina de estados finitos (FSM) es una abstracción matemática que a veces se utiliza para diseñar la lógica digital o programas de ordenador. Se trata de un modelo de comportamiento compuesto de un número finito de estados, transiciones entre esos estados, y las acciones, similar a un gráfico de flujo en el que se puede inspeccionar la forma lógica es cuando se cumplen ciertas condiciones. Tiene una memoria interna limitada, una función de entrada que se lee en una secuencia de símbolos, uno a la vez sin tener que ir hacia atrás, y una función de salida, que puede ser en forma de una interfaz de usuario, una vez que el modelo se implementa. La operación de un FSM comienza a partir de uno de los estados (llamado un estado de inicio), pasa a través de las transiciones en función de la entrada a los diferentes estados y puede terminar en cualquiera de las disponibles, sin embargo, sólo un cierto conjunto de estados marcar un flujo de éxito de la operación (aceptar llamadas de los estados).

Como resultado obtenemos un método fácil de implementar una IA, lo cual es una gran ventaja. Sin embargo las IA desarrolladas mediante maquinas de estado finitos tienden a generar estados para situaciones demasiado concretas, lo cual puede llevar a serios problemas si la IA se acaba convirtiendo en algo demasiado complejo.

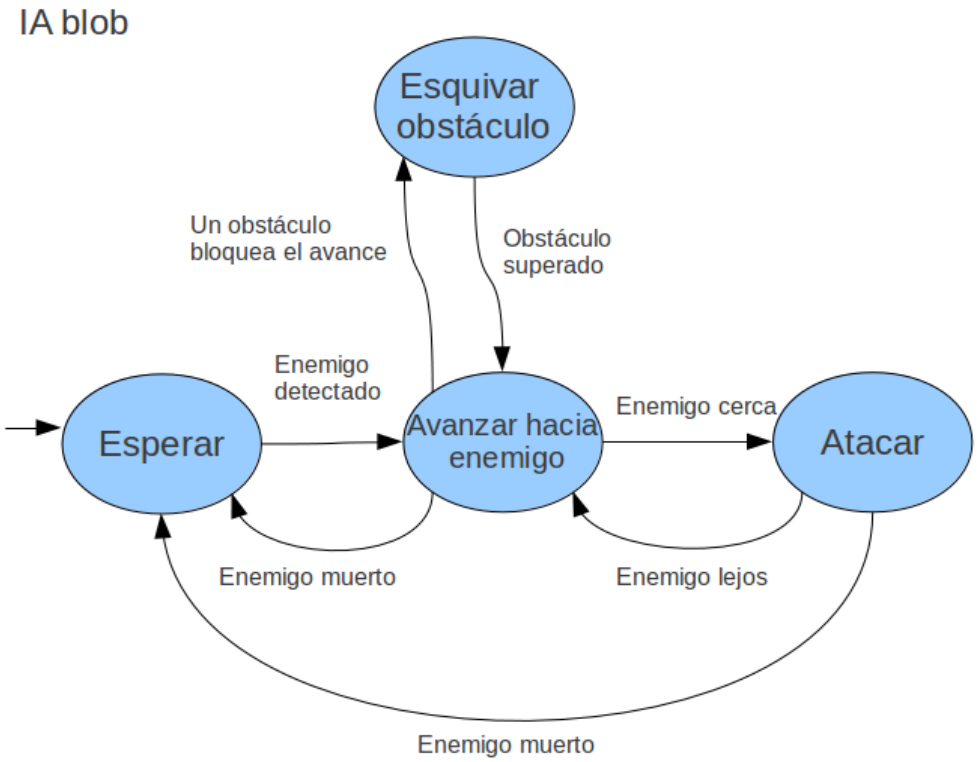
Para solventar el posible problema anteriormente mencionado se usan las maquinas de estados jerárquicos. Dichas maquinas funcionan de forma parecida a las maquinas de estados finitos. La diferencia radica en la posibilidad de crear “superestados” que contengan diferentes estados y sus respectivas transiciones. Sin embargo, como el diseño de las inteligencias va a ser lo más simple posible, no se vio necesaria la utilización de maquinas de estados jerárquicos.

Para obtener un comportamiento menos predecible, se ha utilizado lógica difusa en las transiciones de estados, es decir, las variables de las cuales dependen las transiciones de estados no tiene siempre un valor fijo, sino uno aleatorio dentro de un rango, haciendo que el comportamiento de un agente sea ligeramente diferente al de otro agente del mismo tipo.

4.2.2.1 Blob y comandante blob



Blob es la tropa básica de cuerpo a cuerpo del jugador. Es también el soldado con IA más simple. Su comportamiento puede verse en el siguiente diagrama:

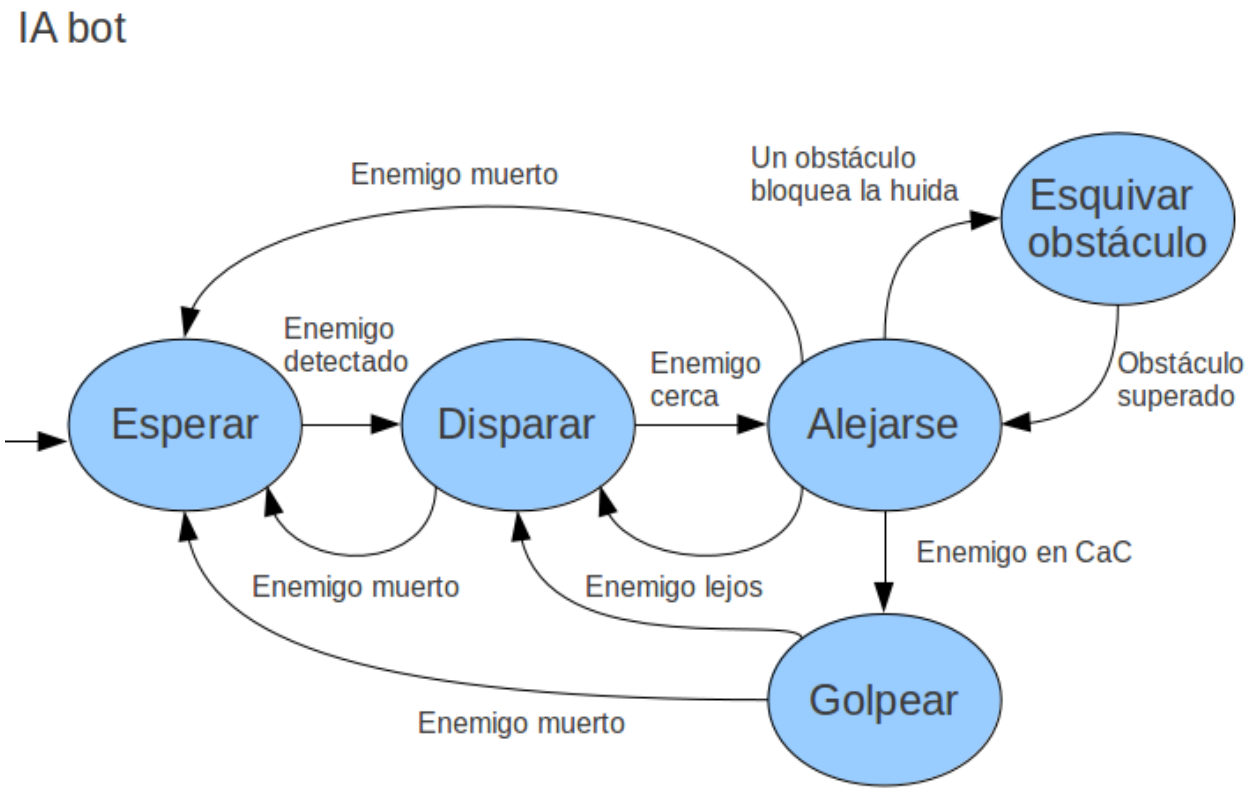


Blob esperará en su posición hasta que detecte un enemigo, entonces avanzará hacia él esquivando los posibles obstáculos que haya en su camino. Una vez que se encuentre a distancia de cuerpo a cuerpo, atacará al enemigo. Si el enemigo se aleja lo perseguirá de nuevo, y si el enemigo muere volverá a su estado inicial de espera.

Su versión comandante tiene un comportamiento idéntico, con la salvedad que la maquina de estados será interrumpida cuando el jugador desee controlar al comandante personalmente.

4.2.2.2 Bot y comandante bot

Los bots son las tropas de largo alcance del jugador. Su comportamiento, aunque simple, tiene algunas diferencias claras con las de sus compañeros de cuerpo a cuerpo:

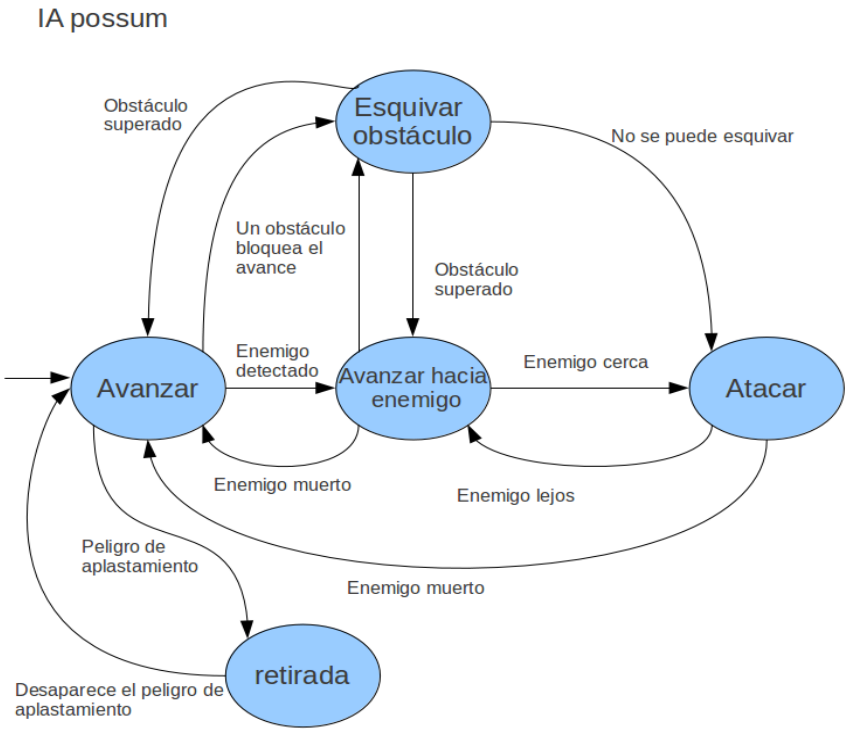


Como puede verse, los bots esperarán en su posiciona hasta que detecten a un enemigo, tras lo cual dispararán hasta darle muerte. Si el enemigo se acerca demasiado intentaran retirarse esquivando los posibles obstáculos que se encuentren. Si el enemigo esta a distancia de cuerpo a cuerpo, no les quedara otra que golpearle. De nuevo, si el enemigo muere vuelven a su estado inicial de espera.

Su versión comandante tiene un comportamiento idéntico, con la salvedad que la maquina de estados será interrumpida cuando el jugador desee controlar al comandante personalmente.

4.2.2.3 Possum y comandante possum

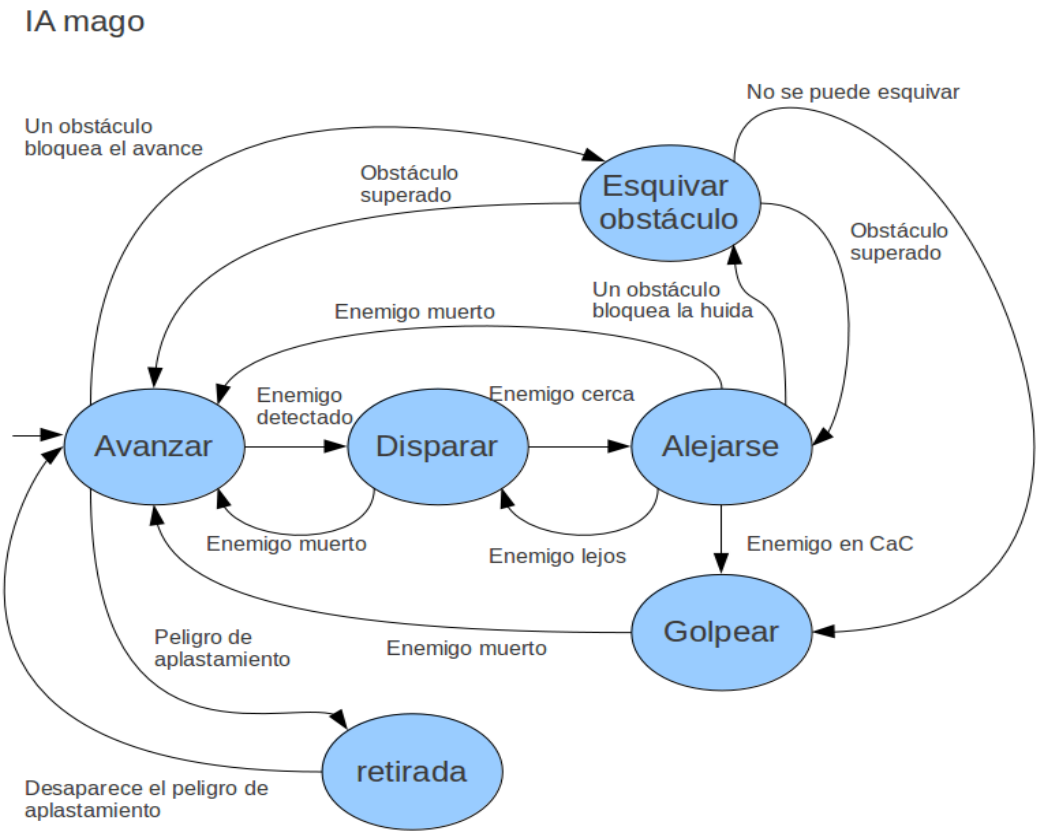
El possum es la tropa básica enemiga, el equivalente al blob. Su comportamiento es muy parecido aunque como puede verse en este diagrama, hay algunas diferencias:



La principal diferencia es el estado inicial, “Avanzar”. El possum avanzará por la pantalla hasta detectar un enemigo, tras lo cual lo perseguirá, evitando todo obstáculo en su camino, hasta poder atacarlo cuerpo a cuerpo. Si el possum no puede esquivar un obstáculo lo intentara destruir. De nuevo, si su objetivo muere, volverá a su estado inicial de a avance. Si, mientras avanza, el possum detecta que un obstáculo le caerá encima, este se retira para esquivarlo.

4.2.2.4 Mago y comandante mago

El mago es la tropa de combate a distancia enemiga. Su comportamiento se asemeja a la de los bots, con algunas diferencias:



Como puede verse, los magos avanzarán hasta que detecten a un enemigo, tras lo cual dispararán hasta darle muerte. Si un enemigo se acerca demasiado intentaran retirarse esquivando los posibles obstáculos que se encuentren. Si el enemigo esta a distancia de cuerpo a cuerpo, no les quedara otra que golpearle. De nuevo, si el enemigo muere vuelven a su estado inicial de espera. Si, mientras avanza, el mago detecta que un obstáculo le caerá encima, este se retira para esquivarlo.

4.2.3 Gráficos

Los gráficos son una de las partes más importantes de los juegos, por el simple hecho de que el jugador recibirá la mayor parte de la información del juego de forma visual.

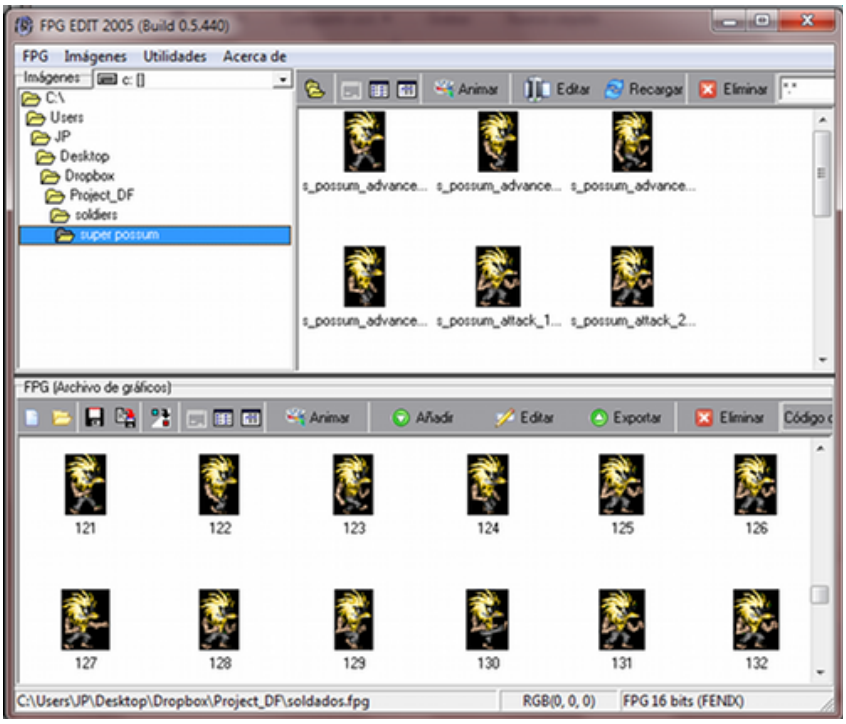
Para el apartado gráfico del proyecto, se decidió aprovechar una gran fuente de recursos que puede ser hallada en Internet, más concretamente la página opengameart.org. Una web donde se ofrece una gran cantidad de contenido, tanto gráfico como sonoro, destinado al desarrollo de videojuegos con licencias libres.

Aunque no todos los gráficos provienen de dicha página, la mayoría, o los más importantes como son las animaciones de los diferentes personajes, si lo hacen.

Sin embargo, todo gráfico debe ser tratado para poder ser usado en el proyecto. En los siguientes apartados se describirá que tratamiento han recibido los gráficos del juego, desde la edición hasta su inclusión en el código fuente.

4.2.3.1 FPG(Fichero para gráficos)

En BennuGD, para un acceso eficiente a las imágenes que usará el juego estas son almacenadas en ficheros fpg. Un FPG no es más que un fichero en el cual se guardan todos los gráficos que van a tener un uso en particular. De esta forma, para este proyecto se usan tres ficheros fpg: Uno para los gráficos de los soldados, otro para los gráficos de los terrenos y un ultimo archivo para los gráficos generales, como los de la interfaz de usuario.



Cabe destacar que dentro del fichero fpg, cada gráfico recibe un número identificativo que sustituye si nombre original. Esto es especialmente útil a la hora de hacer animaciones, como se verá más adelante.

4.2.3.2 Puntos de control

Además de proporcionar un acceso fácil a las imágenes, los ficheros fpg, o más bien el editor de FPGEdit (programa editor de archivos fpg) proporcionan otra herramienta de gran utilidad, los puntos de control.

Los puntos de control son puntos designados dentro de un gráfico por el usuario. Estos resultan de gran utilidad a la hora de centrar los gráficos o determinar, por ejemplo, desde donde debe salir un disparo o donde se encuentra, dentro del gráfico, el arma del personaje.

En el ejemplo de arriba el gráfico del mago tiene tres puntos de control de color blanco. Dichos punto no se verán en el juego pero se utilizaran para saber el centro del gráfico (punto 0, en el pecho del mago), para determinar donde esta la base del gráfico (punto 1, junto al pie del mago), y para determinar donde será creado el disparo del mago cuando este dispare (punto 2, en el bastón del mago).

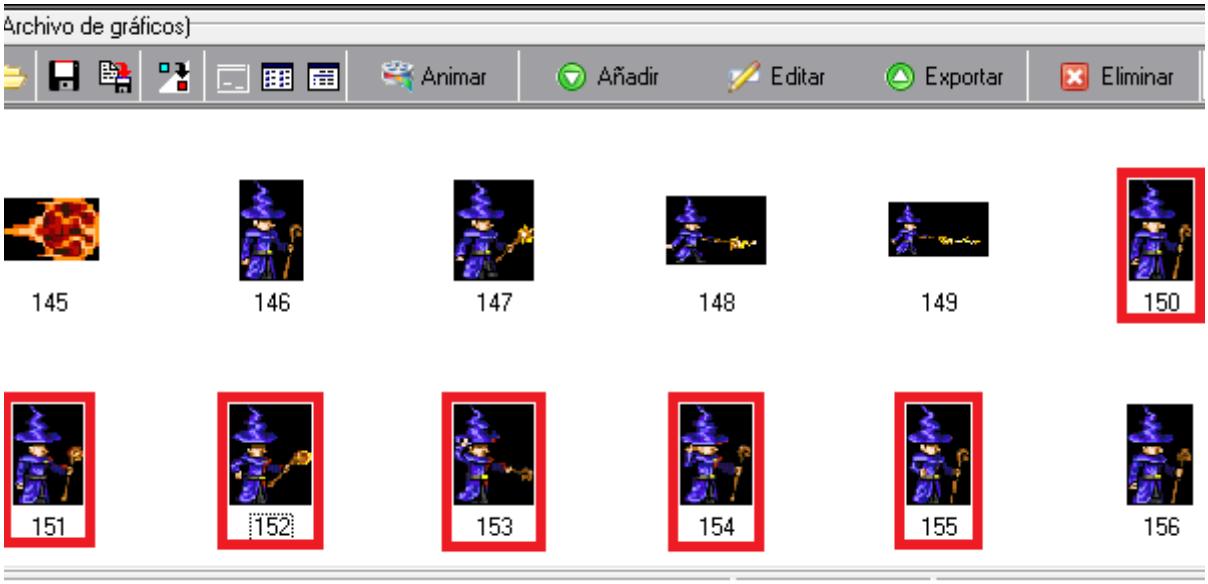
4.2.3.3 Animaciones

Sin duda alguna, el máximo exponente de los gráficos de un videojuego son las animaciones. De estas depende que el aspecto del juego sea fluido y que los personajes parezcan ser algo más que un montón de bits.

El primer paso es el diseño de la animación. Gracias al portal web opengameart.org este paso se hace innecesario. En dicho portal podremos encontrar las animaciones de personajes en este formato:

De aquí tendremos que seleccionar que imágenes queremos y aplicarles un fondo negro (en BennuGD el color negro puro es tratado como transparente) para obtener imágenes como las siguientes:

Estas imágenes serán guardadas en el fichero fpg, tras lo cual se les colocarán los puntos de control necesarios.



Una vez hecho esto, sólo es necesario crear un array en el programa con los números de los gráficos que forman la animación. De esta forma solo hay que recorrer dicho array para crear la animación.

```
int anim_mage_stand[]=156,157,158,157;
int anim_mage_advance[]=135,136,137,138;
int anim_mage_shoot[]= 150,151,152,153,154,155;
int anim_mage_attack[]=139,140,141,142,141,140,139 ;
int anim_mage_magic_attack[]=146,147,148,149,148,147,146;
```

En total se han editado 183 imágenes para su uso en animaciones de soldados.

4.2.4 Sonidos y música

Gracias a la gran variedad de herramientas que proporciona BennuGD, y a la generosidad de los usuarios del portal opengameart.org, este apartado del juego no ha requerido tanto esfuerzo como el apartado gráfico.

De hecho, la única edición que se ha tenido que hacer sobre los diferentes archivos o sonidos ha sido su conversión a un formato de audio apropiado.

En BennuGD el formato más utilizado para los efectos de sonido es el formato WAV. Este formato contiene una grabación digital de audio sin comprimir, y que por tanto ocupa mucho espacio en disco (y en memoria).Por lo tanto, los archivos wav son recomendables sólo cuando se quieren utilizar sonidos realistas pero relativamente cortos de duración, como puede ser una explosión, un choque. Dado que en BennuGD la carga en memoria del archivo wav sólo se realiza una vez, pudiendo después repetir el sonido las veces que queramos, este formato resulta ideal.

En cuanto al formato para los archivos de música existen dos opciones entre las que elegir: formato MIDI o formato OGG. Los archivos MIDI (.mid) ocupan muy poco espacio porque sólo contienen una representación de las notas que componen la música, y es la tarjeta de sonido la que se encarga (con menor o mayor fidelidad según su calidad) de reproducir los instrumentos. Por lo tanto, es el formato ideal si se quiere crear juegos que ocupen lo mínimo posible. Los archivos OGG en cambio, contiene una grabación digital de sonido comprimida al máximo, y están recomendados por su excelente calidad; si se quiere usar música grabada en un juego, es el formato adecuado. El formato escogido pues fue OGG.

5 Instrucciones de uso

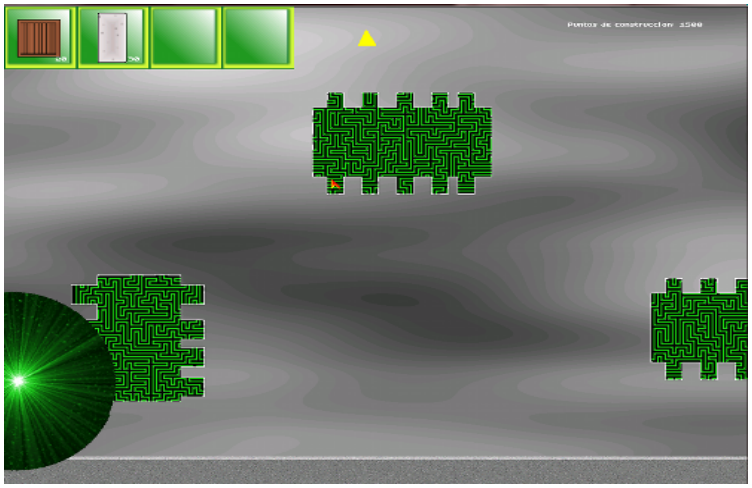
En Project Defense tu objetivo es defenderte del ataque de un ejército enemigo. Para sobrevivir deberás preparar bien tus defensas para repeler el ataque o morir en el intento, ¿serás capaz?

5.1 Menú Principal

Aquí es donde se inicia el juego, las opciones son:

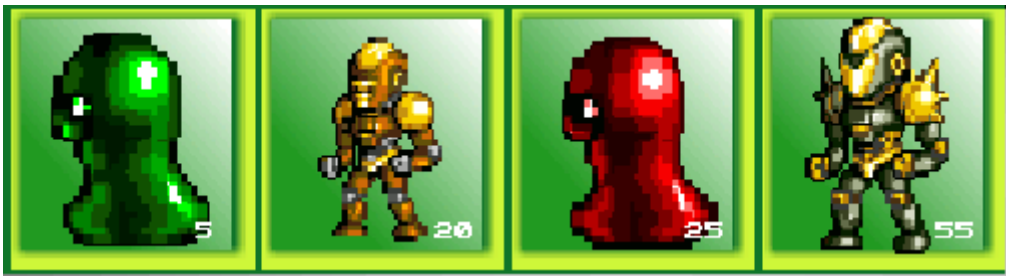
- **Nueva Partida:** empezar una partida nueva.
- **Cargar mapa:** Cargar un mapa previamente creado.
- **Créditos:** Muestra los créditos del juego.
- **Salir:** Salir del juego.

5.2 Modo Preparación



En la parte superior derecha de la pantalla puede verse la cantidad de puntos de construcción de la que dispones. Cada elemento que quieras incluir en tus defensas tiene un coste en puntos. Los puntos que no gastes se sumaran a tu puntuación final.

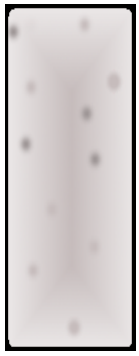
En la parte superior izquierda se pueden ver los elementos que pueden ser usados en la construcción y sus respectivos costes en puntos. La flecha amarilla te permitirá cambiar entre elementos de terreno y soldados.



Las opciones son:

Terrenos

Caja		Es una caja de madera corriente y moliente.

Muro		más alto, caro y resistente que la caja.

Soldados

Blob		Soldado raso de cuerpo a cuerpo de comportamiento simple, cargara contra el enemigo en cuanto lo vea. Barato, útil en grandes números.
Bot		Soldado raso de combate a distancia. Disparará al enemigo cuando este al alcance. Si el enemigo se acerca demasiado se retirará. Sus disparos atraviesan las cajas.
Comandante Blob		Soldado de élite de cuerpo a cuerpo. Mucho más resistente y fuerte que el blob normal. Controlable por el jugador.

Comandante Bot		Soldado de élite de combate a distancia. Mucho más resistente y fuerte que el bot normal. Sus disparos atraviesan todo tipo de terreno. Controlable por el jugador.

Para colocar un elemento haz clic izquierdo sobre el elemento. El elemento aparecerá algo traslúcido, eso significa que de momento no esta incluido y puedes llevarlo a cualquier parte del mapa.

Para colocarlo en el mapa haz clic izquierdo. Ahora el elemento dejara de ser traslúcido y estará preparado para ser colocado en el mapa. Ten en cuenta que ahora el elemento chocará con elementos de terreno o cualquier otra cosa que no pueda atravesar. Para volver al modo traslúcido haz clic derecho.

Finalmente, para colocar el elemento haz clic izquierdo en modo no translucido.

Para mover un elemento ya colocado haga clic derecho sobre el. Dicho elemento se hará traslúcido.
Para eliminar un elemento seleccionado en modo traslúcido haga clic derecho.
En resumen, para avanzar en la creación se usa el clic izquierdo y destruir o retroceder el clic derecho.

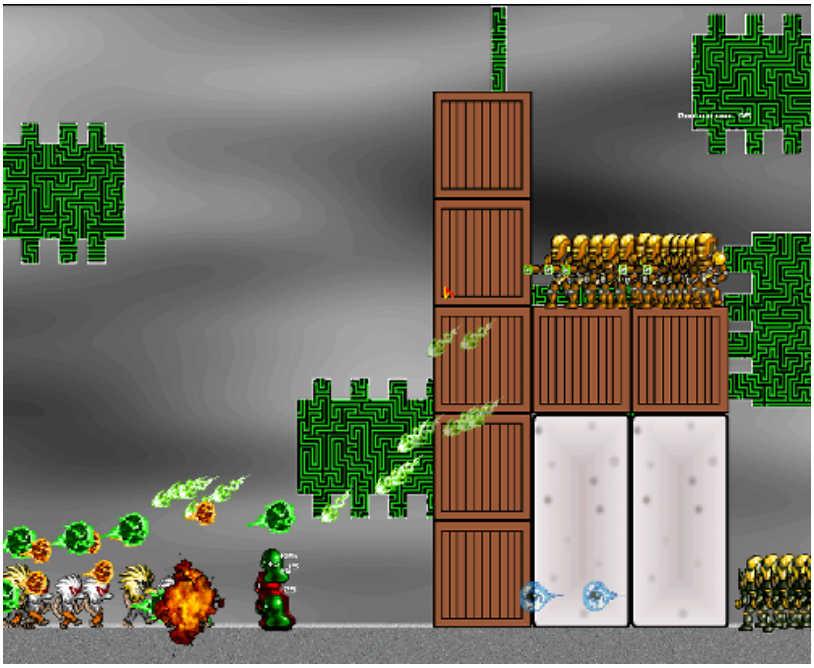
La magia del portal de entrada destruirá cualquier elemento de terreno o soldado que entre en contacto con él! A la hora de preparar tu defensa ten en cuenta que el portal protegerá de los disparos a los enemigos que surjan de el!

Durante la preparación de la defensa pulsa la barra espaciadora para entrar en el menú de pausa, cuyas opciones son:

- **Listo:** Acaba la preparación de las defensas y se pasa al modo batalla.
- **Guardar Mapa:** Guardar el mapa actual.

- **Continuar:** Continuar con la preparación de la defensa
- **Salir:** Salir al menú principal.

5.3 Modo Batalla



El ejército enemigo entrara por el portal en oleadas. Te superan en número así que más te vale haber preparado una buena defensa!

En la parte superior derecha podrás ver tu puntuación actual.

Durante la batalla pulse la barra espaciadora para entrar en el menú de pausa, cuyas opciones son:

- **Continuar:** Continuar con la batalla
- **Salir:** Salir al menú principal.

5.3.1 Personajes controlables

Puedes obtener el control de uno de tus comandantes en cualquier momento de la batalla haciendo clic izquierdo sobre él. Aparecerá una flecha amarilla sobre él para indicar que esta siendo controlado. Sólo puedes controlar a un comandante a la vez y no podrás dejar de controlarlo hasta que muera.

Los controles son los siguientes:

- **A:** Moverse a la izquierda.
- **D:** Moverse a la derecha.
- **W:** Saltar.
- **Clic izquierdo:** Atacar. Atacarás cuerpo a cuerpo con el comandante blob o con el comandante bot si el enemigo esta muy cerca. Mientras ataques cuerpo a cuerpo pulsa S para realizar el ataque secundario. Dispararás con el comandante Bot, apuntando al cursor del ratón.

5.4 Game over

La partida terminará cuando:

- El enemigo llega al final de la pantalla, que es tu ruta de escape. Si el enemigo la alcanza habrás perdido. **[DERROTA]**

- Todos los enemigos han sido derrotados **[VICTORIA]**

Se te mostrará tu puntuación. Para volver al menú principal pulsa ENTER o ESC.

6 Problemas y soluciones

Hay que destacar que a lo largo del desarrollo del proyecto, han surgido problemas de todo tipo, algunos de solución rápida y sencilla y algunos que han planteado serios retos a la hora de solucionarlos. En esta sección describiré los problemas que más han afectado al desarrollo de este proyecto, así como o la solución encontrada.

6.1 Pruebas realizadas

Todas las partes pertenecientes a la aplicación han sido puestas a prueba constantemente. Sin embargo las pruebas más duras se han llevado a cabo al final del desarrollo del mismo. Para ello una serie de versiones beta fueron enviadas a varios voluntarios para que trastearan con ella y detectaran la mayor cantidad posible de errores o partes a mejorar.

A raíz de estos tests se han encontrado y corregido una gran cantidad de errores. Desde los controles o la interfaz hasta comportamientos ilógicos de la IA.

De hecho, las pruebas no han cesado y dudo que lo hagan pues, como ya he mencionado anteriormente, tengo intención de continuar con este proyecto mejorando lo máximo posible.

6.2 Rendimiento, colisiones, y puntos de control

Si en algún punto del desarrollo del proyecto la falta de experiencia en el manejo de las herramientas escogidas ha dado pie a problemas, ha sido aquí. Este es un problema en cascada cuya una fuente es, ni más ni menos, la falta de experiencia.

Por desgracia descubrí la gran utilidad del uso de los puntos de control demasiado tarde. En un principio, para conseguir que una animación quedara centrada, es decir, que el personaje no temblara aun cambiado de postura (o forma en el caso del blob), las imágenes de personajes consistían en grandes recuadros donde el personaje quedaba centrado, teniendo espacio suficiente para la imagen que mayor espacio ocupara en alguna de las animaciones.

Este método resultaba tedioso y consumía una gran cantidad de tiempo animar un solo personaje. Aquí cabe recordar que el lenguaje BennuGD trata el color negro puro como si fuera trasparente, motivo por el cual el resultado final, al menos, era bueno.

Por aquel entonces el método de detección de colisiones utilizado era la función *collision()*. Dicha función devuelve la id de un proceso que estuviera colisionando con el proceso que llama la función. Como parámetro puede especificarse si se busca confirmara la colisión con un proceso en concreto o con cualquier instancia de un tipo de proceso (un enemigo en particular o cualquier enemigo, pro ejemplo).

Tras unas cuantas pruebas descubrí que el rendimiento de la aplicación era realmente bajo y que unos pocos elementos en la pantalla ya hacían temblar la cpu. Tras intentar recortar cálculos de donde se pudiera, descubrí que la función *collision()* hace el calculo a nivel de pixel, por lo cual a parte de la gran precisión, se consigue también el bajo rendimiento obtenido. Había pues que encontrar otro método de detección de colisiones.

BennuGD ofrece dos variantes de la función *collision()*: *collision_box()* y *collision_circle()*. Ambas ofrecen un rendimiento mucho mayor a coste de una perdida en la precisión. La función *collision_box()*, que calculaba la colisión en base a las distancias de los ejes de las imágenes de los diferentes procesos, parecía idónea, salvo por un detalle. Dicha función ignoraba las transparencias.

Las imágenes y la detección de colisiones no eran compatibles, y volver a la colisión por pixel no era una opción. Se intentaron muchas alternativas, entre ellas la idea de crear subprocesos con gráficos rectangulares transparentes que se movieran junto a los personajes y que fueran los encargados de detectar las colisiones fue la que más prometía, pero topé con la dificultad de conseguir una colisión los bastante precisa , pues la coordinación de los subprocesos con los personajes complicaba mucho la lógica del programa. Ademas el número de procesos activos se doblaba, afectado negativamente al rendimiento.

No fue hasta que descubrí y entendí el uso de los puntos que pude dar con una solución satisfactoria. Para más detalles sobre dicha solución no hay más que leer el apartado *4.2.3 Gráficos* de esta memoria. Cabe destacar que para aplicar la solución se tuvieron que reeditar todas las imágenes de soldados del proyecto.

Este problema ralentizó en gran medida el avance del proyecto y es, sin duda alguna, un error que me esforzaré por no repetir en ningún otro proyecto.

6.3 El coste de una caja

Este ha sido uno de esos errores curiosos que , por su naturaleza, requiere mucho más esfuerzo detectarlo que corregirlo.

El error se producía en el Modo Preparación, al añadir un blob al mapa. Al hacerlo, la cantidad de puntos sustraídos a los puntos de construcción equivalía al coste de una caja y no de un blob.

En primer lugar pensé que la llamada a la función que calcula y sustrae los costes en puntos de construcción y que se encontraba dentro del código del proceso elemento() contenía algún error. No era así, todo correcto.

El error pues debía encontrarse en la misma función que calcula y sustrae los puntos de construcción. Ningún error a la vista.

Se revisaron las constantes, todos y cada uno de los procesos relacionados con el Modo Preparación, que no hubiera alguna asignación errónea, algún número equivocado, pero nada.

La función *Control_puntuacion_construccion ()* resta o suma el coste de un elemento a los puntos de construcción. Para calcular cuanto debe sumar o restar se basa en el gráfico que se le pasa como parámetro. Cuando un proceso elemento con el gráfico de un blob llama a esta función, la función calcula que el coste a sumar o restar es el coste de un blob, valor dado por una constante en el programa.

El problema radica en que lo que se pasa por parámetro no es un gráfico, es la referencia a un gráfico, más concretamente el número que ocupa dentro de su fichero fpg. Dentro del archivo fpg llamado “soldados.fpg”, los primeros gráficos

corresponden a todas las animaciones de blob. El gráfico de la caja se encuentra guardado dentro del fpg llamado “Terrenos.fpg”, y resulta que también es el primer elemento. Cuando se le mandaba el gráfico de un blob o una caja a esta función, en realidad se estaba enviando el mismo valor numérico, 1. De modo que la función no sabia distinguir y siempre elegía el mismo resultado.

La solución fue simple. Modificar la función para que tuviera e cuenta a que fichero fpg pertenecía la imagen.

6.4 La partida que nunca acabó

Este error fue especialmente difícil de corregir debido a lo complicado que resultó su reproducción. Durante una de las muchas pruebas que se hicieron una de las partidas no terminó cuando debía. El error no volvió a ocurrir hasta dos meses más tarde, a manos de uno de los voluntarios betatesters.

Intentar reproducir el error era difícil pues no se sabia con exactitud qué lo causaba, ni siquiera que tipo de condiciones, simplemente había una pocas veces en las que la partida no acababa aún habiendo eliminado a todos los enemigos.

De nuevo no parecía haber un error en el código de fin de partida. El problema no era un fallo en el código, sino en el diseño de la IA.

Tras indagar descubrí cual era el problema: Los magos enemigos aparecían por el portal y descubrían que había enemigos demasiado cerca para su gusto, así que retrocedían, sobrepasando el límite de la pantalla, hasta una distancia segura desde la cual disparar. Por su parte los soldados del héroe corrían en la persecución de estos magos, alcanzándolos gracias a su superior velocidad. Sin embargo, para realizar el ataque debían detenerse, por lo que los magos volvían a escapar antes de recibir el golpe. Al no encontrarse nunca en una situación segura los magos seguían retrocediendo y retrocediendo, por lo que no morían y la partida no finalizaba.

La solución más simple fue limitar su huida al límite de la pantalla, de modo que no tendrían su de escapatoria ilimitada.

7 Conclusiones

7.1 Repaso de objetivos

A lo largo del desarrollo de este proyecto ha habido un cambio en cuanto a objetivos y estructura. Aun así creo que los objetivos se han cumplido.

Sin embargo, dado que el resultado ha diferido un poco de la idea original, dando pie a la decisión de mostrar el producto como una versión de demostración, pues creo que aun hay que mejorar varios aspectos del juego antes de lanzar la primera versión.

En cuanto a los objetivos personales, estos se han cumplido con creces. He aprendido a usar nuevas herramientas y tecnologías, he visto los problemas que pueden surgir

en el desarrollo de un proyecto (y como solucionarlos) y he ganado mucha experiencia .

7.2 Valoración personal

Este proyecto ha sido una experiencia muy positiva que me ha permitido aprender mucho sobre el desarrollo de un videojuego. Ha sido todo un reto y creo haber aprendido mucho de los errores cometidos.

También debo admitir que me ha resultado divertido y estimulante trabajar en este tipo de proyecto. Era la toma de contacto que buscaba. Además me ha mostrado lo mucho que me queda por aprender, no solo en cuanto a herramientas de programación, sino más bien en cuanto a la organización y gestión de un proyecto.

7.3 Futuras mejoras

Si hay algo que nunca le sobrará a un videojuego eso son mejoras. Aquí presentaremos las principales mejoras que se planean incluir en el juego en un futuro próximo:

– **Acción en movimiento:** Permitir los disparos y ataques en movimiento dará una mayor sensación de fluidez al juego, un mayor dinamismo. Esta mejora no ha sido implantada aún debido a la inexistencia de las animaciones necesarias. Será una de las primeras mejoras a implantar.

– **Variedad de opciones:** Incluir una mayor variedad de terrenos, tropas he incluso nuevos mecanismos como trampas. En la variedad esta el gusto y es justo lo que este proyecto necesita.

– **Mejoras de mecánicas:** Tanto la IA como las mecánicas de saltos, física y control de personajes pueden (y deben) ser mejoradas. Se plantea la inclusión de librerías de físicas 2D.

– **Opciones de configuración:** Algo que falta en la versión actual es la posibilidad de configurar el juego. En un futuro se le permitirá al jugador establecer la cantidad de puntos de construcción, cambiar el volumen de la música y los efectos de sonido, jugar en modo ventana o a pantalla completa, etc...

Hay otras muchas ideas y formas de mejorar el proyecto, de modo que se espera que este vaya mejorando poco a poco.

8 Bibliografía

Libros:

- Stuart Russell y Peter Novig: **Inteligencia Artificial: Un Enfoque Moderno**. Pearson Education.

- Oscar Torrente Artero. **Curso de iniciación a la programación de videojuegos con el Lenguaje BennuGD.**

páginas web:

- [www.bennugd.org] página oficial de BennuGD

- [http://opengameart.org] página que reúne gran cantidad de material gráfico y sonoro de licencia libre destinado al desarrollo de videojuegos.

Firmado: Juan Pedro Rodríguez Extremera