Hito2
Proyecto: Los osos amorosos

Francisco Andrés Sanchís Pérez
Juan Rubén Segovia Vilchez
Joaquín Vegara García

1. Ámbito del juego

El proyecto consistirá en la realización de un juego de estrategia 3D en tiempo real (tipo warcraft o Age of Empires) utilizando la librería AllegroGL. La finalidad del juego es derrotar a los equipos enemigos mediante la destrucción de todas sus unidades usando tu propio ejército y ayudado de los recursos del juego. El entorno del juego estará ambientado en la época actual, por lo tanto se empleará la tecnología desarrollada hasta la fecha. Además se dispondrá de unidades civiles que se ocupen de las tareas de mantenimiento y administración de los recursos que se ponen en el mapa.

2. Sector al que va dirigido el juego

Este juego va dirigido al público en general. No tiene una violencia excesiva que lo haga no apto para menores, aunque por su caracter bélico no se recomienda para menores de 7 años.

3. Requerimientos

3.1 Requerimientos hardware

3.1.1 Requerimientos hardware recomendados

Ordenador PC compatible, tarjeta gráfica con soporte openGL que soporte una resolución de 1024x768, ratón, teclado, monitor, altavoces, tarjeta de sonido.

3.1.2 Requerimientos hardware mínimos

Ordenador PC compatible, tarjeta gráfica con soporte openGL que soporte una resolución de 1024x768, ratón, teclado, monitor.

3.2 Requerimientos software

Windows 98, Me, XP, drivers actualizados de la tarjeta gráfica,

4. Herramientas a utilizar

Dev-C++ para compilar el proyecto.

Allegro y AllegroGL para soporte de rutinas de bajo nivel. (Graficos, sonido, input)

Lib3ds y Cal3D para la carga y visualización de modelos y modelos animados de 3D studio.

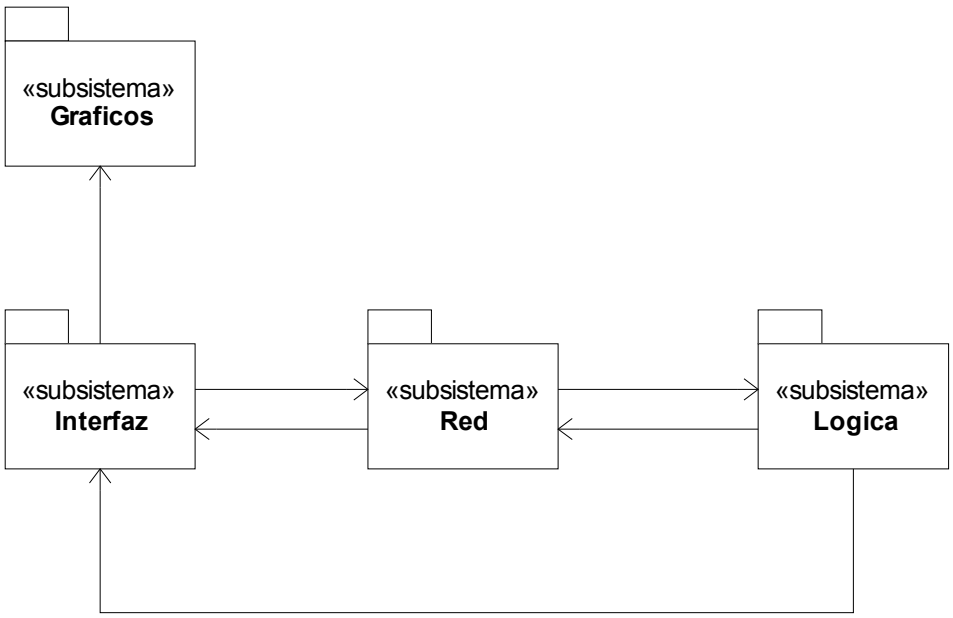
3D studio para modelización de unidades, edificios, etc.

Gimp y Photoshop para el diseño de las texturas e imágenes.

5. Arquitectura del juego

5.1 División modular

El proyecto se divide en varios módulos que realizan tareas diferentes. Se tiene el módulo de graficos, el módulo de interfaz, el módulo de lógica y el módulo de red. En el código, cada módulo se define en un namespace propio, y los ficheros de dicho modulo empiezan todos por una letra minúscula que identifica el módulo. Además cada módulo proporciona una clase llamada Api para su interrelación con el resto de módulos.



En el esquema anterior se observa que el modulo de red hace de comunicador entre el módulo de lógica y el módulo de interfaz. Aunque se verá más adelante es conveniente destacar ahora que el módulo de red no implica una conexión de red, sino que sirve de comunicador abstracto entre lógica e interfaz, permitiendo que las diferencias entre una partida local, y una partida de red queden generalizadas por completo dentro de dicho módulo sin tener que modificar el codigo externo.

El módulo de gráficos proporciona una Api amigable para la creación y manipulación de elementos 2D de GUI, los cuales utiliza el módulo de interfaz. Asimismo proporciona métodos para la visualización 3D como el control de la camara, carga del terreno, modelos, y visualización de objetos que son igualmente utilizados por el módulo de interfaz para visualizar el área de juego.

El módulo de interfaz, muestra los menús del juego, y en su momento, según se inicie una partida local, o en red, inicializará el sistema de red inidicándole si va a ser una partida local o en red. Además, si va a ser una partida local, o se va a ser el servidor de una partida en red, se inicializa también el módulo de lógica.

Es el módulo de lógica el que guarda información sobre los jugadores existentes en la partida, y todas sus unidades, simulando el mundo, y enviando notificaciones de cambios al módulo de interfaz para que se encargue de su visualización.

Toda la comunicación Interfaz – Lógica se realiza a traves del módulo de Red, salvo unas pocas excepciones, en las que el módulo de lógica accede directamente a la Api del módulo de Interfaz, para realizar ciertas consultas (consultas de configuración del juego, o cosas como pedir que se calcule si un disparo impacta a alguna unidad), son en definitiva todas aquellas consultas que la Logica tenga que realizar a la Interfaz.

5.1.1 Interfaz

Dentro de este espacio de nombres se encuentran las clases que gestionan la interacción del usuario con el juego, por lo tanto es la encargada de manejar los periféricos para comunicar las acciones del usuario al módulo gráfico y al módulo de la lógica del juego.

Dentro de este conjunto de clases podemos distinguir: Gestión del panel, interfaz para los menús, interfaz para las unidades, interfaz para la cámara, interfaz del área de juego, configuración e información sobre el mapa.

5.1.1.1 Panel

Son las clases encargadas de gestionar el panel con el que interaccionamos con las unidades del juego y el mapa. Además permite el acceso rápido a acciones usando métodos abreviados del teclado. Se compone de las siguientes clases:

- ***InterfazPanel*** (*iInterfazPanel.cpp* y *iInterfazPanel.h*).

Crea el panel mediante composición de las siguientes clases proporcionando a su vez un interfaz para interaccionar con ellas.

- ***IntMinimapa*** (*iIntMinimapa.cpp* y *iIntMinimapa.h*).

Implementa un minimapa que permite un movimiento más rápido por el mapa haciendo click sobre él.

- ***IntSeleccion*** (*iIntSeleccion.cpp* y *iIntSeleccion.h*).

Implementa el interfaz encargado de mostrar la selección múltiple. Al seleccionar varias unidades se mostrara una lista de éstas en el panel pudiéndose seleccionar desde éste mismo.

- ***IntUnidad*** (*iIntUnidad.cpp* y *iIntUnidad.h*).

Esta clase se encarga de mostrar la unidad seleccionada mediante selección simple. Cuando se realice una selección múltiple también aparecerá una unidad seleccionada por defecto que puede ser la primera de la selección.

- ***IntAcciones*** (*iIntAcciones.cpp* y *iIntAcciones.h*).

Gestiona las acciones disponibles para la unidad seleccionada mostrándolas mediante botones. Cuando se produzca una selección múltiple sólo se mostrarán las acciones comunes del conjunto de unidades seleccionadas. También es posible activar estas acciones mediante atajos de teclado.

5.1.1.2 Menu

Engloba todas las clases que forman los menús del juego. Partimos de una clase base que nos proporciona cinco botones de los cuales sólo mostramos los que nos interesen. El diseño de esta clase permite organizar menús mediante una jerarquía de padres e hijos facilitando la gestión de estos.

- ***MenuPrincipal*** (iMenuPrincipal.cpp y iMenuPrincipal.h).

Esta clase representa el nivel más alto de la jerarquía de menús (no tiene padre). Muestra un menú con 5 opciones: *Crear Partida*, *Unirse a Partida*, *Configuración*, *Créditos* y *Salir*.

- ***MenuNombre*** (iMenuNombre.cpp y iMenuNombre.h).

Muestra un menú con las opciones *Aceptar* y *Volver* y un *textbox* en el que es posible introducir el nombre del jugador.

- ***MenuMapa*** (iMenuMapa.cpp y iMenuMapa.h).

Muestra un menú con las opciones *Crear Partida* y *Volver* además de un *listbox* donde aparecen todos los mapas disponibles permitiendo la selección de cualquiera de ellos y mostrando el

preview del elegido.

- ***MenuPartida*** (iMenuPartida.cpp y iMenuPartida.h).

  Muestra un menú con las opciones *Empezar Partida* y *Volver.*

- ***MenuUnirse*** (iMenuUnirse.cpp y iMenuUnirse.h).

Muestra un menú con las opciones *Aceptar* y *Volver* y un *textbox* en el que será posible introducir la dirección *ip* de la máquina a la que nos conectaremos para las partidas en red.

- ***MenuConfiguracion*** (iMenuConfiguracion.cpp y iMenuConfiguracion.h).

  Muestra un menú con las opciones *Configuracion*, *Video*, *Audio*, *Juego* y *Volver.*

5.1.1.3 Unidades

Engloba las clases encargadas de visualizar las unidades en pantalla y gestionar los elementos de interfaz para interrelacionarse con dichas unidades. Tenemos una clase base abstracta *Unidad* el la que generalizamos todos los tipos de unidades proporcionando un interfaz común en el que permitimos acciones como cambiar la posición o el ángulo de las unidades. Las unidades a su vez las clasificamos como unidades seleccionables (edificios, humanos, vehículos, ...) y unidades no seleccionables (árboles, decorado, misiles, ...). Las unidades seleccionables las diferenciamos a su vez en unidades uniseleccionables (edificios) y unidades multiseleccionables (resto de unidades).

5.1.1.4 Camara

La cámara está implementada en la clase *Camara* definida en los archivos *iCamara.cpp* y *iCamara.h*. Esta clase realiza el movimiento de la cámara mediante un movimiento uniformemente acelerado en cada dimensión, hasta la mitad del recorrido, momento en el cual el movimiento es decelerado para alcanzar el punto final suavemente. Por esto, para un movimiento sencillo, la función de posición respecto al tiempo tiene la forma de medio periodo de una función sinusoidal.

Puesto que se puede cambiar la posición deseada antes de alcanzar la posición deseada anterior, en realidad lo que se hace es empezar a desacelerar el movimiento a partir de cuando se calcula que actuando así la cámara va a llegar a la posición deseada, para esto partiendo de la ecuación del movimiento uniformemente acelerado, se tiene que cumplir que la ecuación tenga una o dos soluciones, es decir, el discriminante (argumento de la raíz cuadrada) sea mayor o igual a 0, ya que no tiene sentido físico la solución compleja.

El discriminante es *(vel^2)-2*acel*(posIni-posFinal)*.

La inclinación se trata con el mismo procedimiento.

5.1.1.5 Configuracion

La clase *Configuración* está implementada en los archivos *iConfiguracion.cpp* y *iConfiguracion.h* y nos proporciona unos métodos para poder configurar los parámetros de la aplicación como pueden ser el directorio principal, el nombre del jugador o la dirección *ip* del servidor.

5.1.1.6 Información del mapa

La clase *iInfoMapa* definida en los archivos *iInfoMapa.cpp* y *iInfoMapa.h* proporciona las herramientas necesarias para obtener la información de un mapa descrito en un archivo *.map*. Estos archivos contienen la siguiente información:

- Nombre del mapa.
- Directorio del mapa.

- Número máximo de jugadores.
- Descripción del mapa.

5.1.1.7 Interfaz del juego

La clase *Juego* esta definida en los archivos *iJuego.cpp* y *iJuego.h*. Esta clase hereda de la clase *Menu* para poder volver al menú principal de forma transparente sólo con establecer como menú actual el menú padre desde donde se entró. Esta forma de diseño también nos permitirá en un futuro incluir un menú que sea útil durante el juego.

La clase *Juego* es la encargada de crear la cámara, cargar el terreno, cargar el panel, crear unidades y todos los elementos visualizables en el mapa. También explora los periféricos de entrada para poder interactuar con el juego.

5.1.1.8 Api

Proporciona por un lado métodos de inicialización llamado directamente por *main.cpp* para inicializar todos los aspectos visuales del juego. También proporciona unos cuantos métodos de consulta que permiten obtener información sobre el módulo de interfaz a clases de la propia interfaz y de la lógica. Esta clase ofrece también métodos para recibir notificaciones del cambio del estado del mundo.

5.1.2 Gráficos

Este módulo proporciona una api sencilla e intuitiva al modulo de interfaz simplificandole sus tareas, de modo que el módulo de interfaz se pueda centrar en la "lógica subyacente a la interfaz" mientras este módulo se encarga de las rutinas de visualización de bajo nivel (allegro y opengl)

5.1.2.1 Imagen

Las imagenes son uno de los elementos básicos del módulo de graficos, representa una imagen que puede ser pintada en la pantalla. No obstante, la api del módulo graficos no permite pintar imagenes en pantalla directamente, sólo cargar y liberar imagenes.

5.1.2.2 Entidad2D

Esta clase es la clase base de todas las entidades 2D pintadas en la pantalla (sprites, text boxs, botones, etc), almacena las coordenadas de la entidad, en especial la coordenada z, que indica que objetos 2D se pintan encima de otros. Tiene un control automático de todas las entidades2D creadas, y provee de un método estático para pintarlas todas, por orden de coordenada z. Para ello recorre una lista dinámica ordenada por z, que contiene un puntero a cada Entidad2D creada, y llama al método Pintar de cada Entidad2D. Cada clase derivada redefine el método pintar.

Una entidad 2D puede ser desactivada, la entidad sigue existiendo y almacenando sus valores, pero no es pintada en pantalla. En cualquier momento puede volver a ser activada.

La api del módulo gráfico no permite la creación directa de Entidades 2D.

5.1.2.3 Sprite

Un sprite es una entidad2D al que en el momento de construirlo se le pasa un puntero a una Imagen. El sprite, redefine el método Pintar de Entidad2D para hacer que aparezca en pantalla la imagen con la que ha sido construido en las coordenadas en las que se encuentra el sprite.

La api del módulo gráfico permite la creación y manipulación de sprites, tales como modificar su posición, o activar o desactivar el sprite.

5.1.2.4 Boton

Un boton es una entidad2D que se crea a partir de una imagen del boton, una imagen de selección, una fuente de texto, y un texto. El boton pinta la imagen de fondo, y el texto escrito con la fuente de texto pasada en el centro de la imagen. Cuando el raton pasa por encima de la imagen, pinta ademas la imagen de selección al 50% de opacidad. Si además de esto se pulsa el boton del raton se eleva el nivel de opacidad al 75%.

Además se le puede indicar al boton que avise de evento de click, pasandole un puntero a bool, que el propio boton pondrá a true cuando se produzca un click, y a false cuando pase el evento.

5.1.2.5 TextBox

Un textbox es una entidad2D, que tiene una imagen de fondo, y una fuente de texto. El textbox muestra un texto encima de la imagen de fondo, y permite la activación de dicho textbox con el raton y la modificación del texto que aparece leyendo el teclado.

La api proporciona además de los habituales, metodos para establecer y obtener el texto actualmente mostrado en un textbox.

5.1.2.6 ListBox

Un listbox permite la selección entre varias opciones textuales. Se crea a partir de una imagen de fondo, uma imagen de opcion seleccionada, y una imagen de seleccion, además de una fuente de texto que se utilizará para escribir las opciones. La imagen de opcion seleccionada se pinta debajo del texto seleccionado, y la imagen de seleccion se pinta al 50% de opacidad encima del

texto que se seleccionaría si se pulsara el boton del raton.

La api proporciona metodos para añadir nuevas opciones y para obtener el número y texto de la opción seleccionada. Además se puede pedir que informe del evento de opción cambiada pasando un puntero a bool. También se le puede pasar un puntero a int para que mantenga actualizado el indice de la opción seleccionada.

5.1.2.7 Fuente

La clase fuente representa una fuente de texto. La api permite cargar y liberar fuentes de texto, más alla de esto no se permite ninguna acción directa sobre una fuente. Si se quiere mostrar un texto en pantalla, la forma de hacerlo es crear un objeto Imagen con el texto, para esto la api permite crear una Imagen a partir de una fuente de texto y un string con el texto a mostrar. Una vez que se tiene la imagen creada se puede crear un sprite que muestre dicha imagen en pantalla.

Además, elementos como el boton, textbox, listbox, ..., crean internamente objetos Imagen con este procedimiento.

5.1.2.8 Modelo

Un modelo es la analogía 3D de una imagen. Es decir, es aquello que puede ser pintado en pantalla. Se puede crear un modelo pasandole un string con el nombre del fichero que contiene la información del modelo. También se puede crear un modelo pasandole un identificador de CallList de openGl, esto lo utiliza por ejemplo la clase Terreno, que genera una CallLists con la información del terreno y crea modelos pasandole los identificadores de dichos calllists.

La api del módulo de gráficos permite crear un modelo a partir de un fichero 3ds, o crear un bitmap 3D, o quad, para poder pintarlo en 3D (esto se hace por ejemplo para pintar los círculos de selección de las unidades)

5.1.2.9 Entidad3D

Una Entidad3D es la clase base de todos los objetos 3D que son pintados en la pantalla. Almacena la posición de la entidad, así como parámetros de control como si está activada (si se debe pintar o no en la pantalla), si es transparente, o si es siempre visible. El hecho de saber si una entidad 3D es transparente o siempre visible a nivel de entidad 3D es necesario ya que las entidades 3D deben pintarse primero las que no son ni transparentes ni siempre visibles, después las transparentes pero no siempre visibles, y por último, las siempre visibles con el DEPTH_TEST desactivado.

La clase Entidad3D proporciona un método PintarTodos igual que hacía Entidad2D, peró resulta de mayor utilidad el método PintarTodosEn que recibe las coordenadas de un rectangulo en el plano y=0, y realiza un clip de todos aquellos objetos que no sean interiores a dicho rectangulo tras aplicacar una proyección con Base:=(y=0) y Direccion:=(x=0, z=0), es decir, todos aquellos objetos cuyas coordenadas x,z estén dentro del rectangulo especificado.

5.1.2.10 Objeto3D

Un objeto 3D representa un modelo que está en algún sitio (es la analogía 3D de sprite), se construye pasandole un modelo, que será el que pintará. Redefine el método pintar de la entidad 3D para adecuar la matriz MODELVIEW de forma que el objeto se situe en las coordenadas y orientación correcta.

5.1.2.11 Terreno

La clase terreno soporta la creación del terreno de juego, y metodos de consulta como obtener la altura de una posición. Para crear el terreno lo divide en sectores y genera una calllist para cada sector; tras esto genera un modelo para cada sector, y un objeto3D para cada modelo. Además de esto genera un modelo y un objeto 3d para el agua.

5.1.3 Logica

Este módulo se encarga de la actualización del Juego, esto comprende mantener la información referente a cada unidad lógica del juego, las unidades se representan mediante una jerarquía de clases que agrupan sus atributos comunes, para cada unidad puede existir una unidad de Inteligencia Artificial que controle su interacción con el resto de unidades e implemente su comportamiento. Para la interacción de este módulo con el resto del juego así como la interacción entre partes de este módulo se proporciona un API, que se implementa como un interfaz de funciones útiles accesibles estáticamente por todos los módulos y que encapsulan los datos internos de la aplicación.

5.1.3.1 Api

La API de la lógica esta definida en los ficheros lAPI.h y lAPI.cpp, se define mediante una clase con métodos y datos estáticos ya que no se van a instanciar objetos de la API, define funciones para diversas partes:

- *Abstracción de jugadores:*

Esta parte define un mecanismo para poder enviar comandos a la lógica desde un jugador abstracto, que sirva en un futuro para Humano o IA.

Mediante las funciones EnviarComando, EnviarParametro, FinComando, se enviaran comandos, que serán interpretadas por las unidades como acciones a realizar (El diseño actual del paso de comandos es temporal ya que cambiará cuando se implemente el Protocolo)

- *Inicialización:*

Aquí se definen las funciones necesarias que se llamarán cuando la lógica comience a funcionar, actualmente Inicializar() prepara la lógica y todos los datos para comenzar a crear unidades y EmpezarPartida() inicializará además parámetros de red.

- Gestión unidades:

Se han implementado dos métodos, Crearunidad y Destruirunidad, que se ocupan de instanciar y guardar los objetos en la Lógica, para que esa unidad empiece a pintarse o deje de hacerlo, es necesario notificar al módulo gráfico de ello.

- Interacción con unidades:

Desde dentro de una unidad hace falta comunicarse con el resto de unidades del juego, para ellos se proporcionan unas funciones que permitirán obtener o cambiar datos de éstas.

Será necesario saber la posición o las dimensiones de otras unidades, modificar la vida de los enemigos, y obtener cualquier otra característica de la unidad.

También se proporcionan funciones útiles, como son buscar unidades en un radio o detectar zonas no accesibles en el mapa.

- Actualizar:

Método encargado de recoger los comandos enviados a las unidades (temporal) y de recorrer las unidades para actualizarlas una a una.

5.1.3.2 Unidades

Jerarquía de clases de unidades, definen los parámetros de éstas y se asocian en algunos casos con una IA para comportamientos que no dependen del jugador.

- Unidad:

Clase base de todas las unidades, define propiedades como posición, velocidad, ángulo, bando, raza, equipo, y métodos para acceder y modificar éstas propiedades.

Métodos para asociar y quitar la IA de una unidad, crea una instancia del tipo que se pase por parámetro y la asocia.
Métodos para establecer y obtener las acciones a realizar por la IA, setNuevaAcción, getAccionActual
Método virtual Actualizar que será implementado por las unidades para las que sea necesario.
Método ActualizaIAUnidad que encapsula el método Actualizar de la IA.

- UnidadSoldado:

Clase que deriva de unidad y que representa a un soldado, define nuevas propiedades características de este objeto como la experiencia.
El constructor de unidades soldado la asigna a una IA mediante el método de la clase base asociaIA y establece su posición inicial y su velocidad.
Método actualizar, llama a intervalos regulares a ActualizarIAUnidad de la clase base para que actualice la IA y basándose en los resultados devueltos de velocidad y ángulo, calcula una nueva posición teniendo en cuenta características físicas de la unidad, por ejemplo, si la unidad tiene poca vida la velocidad devuelta por la IA podrá reducirse.

- Unidad Recurso:

Unidad sin IA que almacena un valor, y representa una fuente de algún recurso del juego que será explotado a lo largo del transcurso del mismo.

5.1.3.3 IA's de unidad

Parte encargada de controlar los comportamientos de las unidades, las clases que se asocien a alguna unidad implementarán un autómata de estados finitos en el método principal Actualizar, que será el que realice transiciones de estados, y en definitiva el que modifique el comportamiento de la unidad.

- IAUnidad:

Es la clase base de las IA's de unidad. Define parámetros básicos necesarios por toda la jerarquía, como son, estado, acción, velocidad, ángulo, o unidad, que será la referencia a la unidad que controlan.

Método setNuevaAcción, será el encargado de cambiar la acción actual.

Cada IA define e implementa unas acciones concretas.

- IAUnidadMóvil:

Es una IA que está pensada para servir a IA's derivadas, implementando para ello acciones primitivas que tengan que ver con el movimiento como PasoAndar, PasoCorrer, PasoParar , que avanzan un paso el movimiento de la unidad, también se implementan eventos de movimiento, como eventoHaLlegado o eventoColision, que serán utilizados por los autómatas de las IA's derivadas para cambiar de estado.

Método setNuevaAcción, se redefine en esta clase para que tenga en cuenta las acciones ANDAR,CORRER y PARAR.

Métodos setParamsMovil, se utilizan para pasar los parámetros de las acciones con movimiento a las IA's (temporal)

- IAUnidadOfensiva:

Clase que implementa las funciones auxiliares, acciones primitivas y eventos necesarios para el combate.

Métodos auxiliares:

AuxBuscarMejorEnemigo: Intenta buscar el enemigo más cercano y más alineado con la unidad actual (que requiera menos giro) en un radio determinado (zona audible + zona visible), si lo encuentra devuelve el identificador de la unidad, sino -1

AuxHayObstaculos: Comprueba si entre la unidad actual y entre la que se pasa como parámetro hay obstáculos de mapa o de unidades grandes, como edificios.

AuxHayEdificios: Busca edificios en la línea que se pasa como parámetro.

AuxEsVisiblePorUnidad: Comprueba si no hay obstáculos (utilizando funciones anteriores) y si además la distancia y orientación de la unidad permite visualizarla.

Hay eventos que se deberán gestionar como, visibilidad de enemigos, cercanía, saber si ha muerto, si la unidad está lejos de la base o si tiene poca vida (No están todos implementados).

Las acciones primitivas definidas en la IA ofensiva, son PasoDisparar que dispara un arma de fuego y calcula un daño al enemigo y PasoAtacar, que calcula un daño cuerpo a cuerpo.

El método Actualizar implementa un autómata.

La jerarquía de las IA's aún está en fase de diseño y es posible que cambie en un futuro, se ha pensado en utilizar herencia múltiple, para por ejemplo definir IA's de unidades que tengan movimiento y sean ofensivas (soldados), y poder distinguir entre otras que sólo tienen movimiento (ingeniero) o que sólo son ofensivas (torretas de vigilancia)

5.1.3.4 Utilidades

- FrecEv:

Esta clase se utiliza para medir frecuencias de ejecución, se define pasando al constructor un porcentaje de ejecución (de 0 a 1) y se llama al método comprobar() que sólo devolverá true ese porcentaje de veces, se utiliza para controlar las ejecuciones de algunos eventos de IA que pueden ser muy costosos, como búsqueda de caminos o de unidades

- FrecReal:

Se basa en la misma idea que la anterior, pero esta vez lo que se mide son frecuencias en tiempo real, es decir, se define pasando un número de veces por segundo, y cuando se llame al método comprobar() éste sólo devolverá true a intervalos regulares. Se utilizará por ejemplo para controlar e independizar las animaciones y la velocidad del juego de la CPU en la que se ejecuta.

- Posición:

Clase que define e implementa un vector en dos dimensiones y las operaciones básicas sobre él, como sumas, restas, producto escalar y módulo.

6. Seguimiento de objetivos

Leyenda:

Objetivo cumplido.

Objetivo en progreso.

Objetivo retrasado.

Objetivo cambiado.

6.1 Hito 1

Fecha de revisión: 06-04-05.

Porcentaje a realizar: 30% (30% Realizado)

6.1.1 Graficos

1. Carga de recursos (modelos, texturas, bitmaps, etc).
   - La tarea fundamental aquí, y la que más horas llevará es la carga de modelos 3D. La carga del resto de recursos resulta trivial con las funciones que proporciona Allegro. (15 horas)
   - Carga de bitmaps, fuentes y sonidos (1 hora)
2. Creación de elementos de GUI utilizables de forma sencilla por el modulo de interfaz. (10 horas)
3. Instanciación de unidades. (30 min)
4. Carga y visualización del mapa. (2 horas)
5. Control de la cámara. (2 horas)

6.1.2 Interfaz

1. Soporte de desplazamiento por el mapa. (1 hora)
2. Inclinación de la vista. (30 min)
3. Realizar menús (5 horas)
4. Interacción con unidades a traves de botones de la interfaz
   - Interacciones con el terreno: (3 horas)
     Movimiento agresivo
     Mover
   - Interacciones con otras unidades: (4 horas)
     A: Atacar unidad
   - Ordenes inmediatas:
     Detenerse
5. Interacción básica con ratón y teclado.
   - Desplazamiento del mapa con el raton (30 min)
   - Búsqueda de unidades con teclas estableclidas (H -> HeadQuarters) (3 horas)
     H: Cuartel general
     . (punto) : Siguiente unidad
   - Asignación de ordenes con teclas establecidas (A -> Atacar) (5 horas)
     A: Atacar
     D: Detenerse
     M: Mover
   - Selección de unidades (2 horas)
   - Selección multiple de unidades (con la tecla shift) (5 horas)

Este punto presenta una mayor complejidad puesto que hay que adaptar la interfaz, de modo que las ordenes que se den, sean dadas a todas las unidades (todas las unidades que puedan realizar dicha orden)

6.1.3 Logica

1. Crear jerarquía básica de clases para las unidades (soldado, edificio, etc). (5 horas)
2. Crear jerarquía básica de clases para la Inteligencia Artificial de las distintas unidades (guerrero, misil, recolector, etc). (5 horas)
3. Implementación de un protocolo básico de comunicación con el módulo de GUI y el Gráfico. (5 horas)
4. Implementar algunas órdenes básicas. (2 horas)
5. Implementar comportamientos básicos (andar, pararse, crear y destruir unidades). (10 horas)

6.2 Hito 2

Fecha de revisión: 19-05-05.

Porcentaje a realizar: 50% (80% Realizado)

6.2.1 Graficos

1. Finalizar el motor gráfico.
   - Carga de modelos con texturas (20 horas)
   - Carga de modelos con compuestos (20 horas)
   - Carga de modelos con partes de color de bando (5 horas)
   - Texturizacion del agua (1 hora)
   - Ligero movimiento del agua (30 min)
2. Soporte de sonido. (3 horas)

6.2.2 Interfaz

- Soporte completo del interfaz.
  - Finalizar botones de ordenes (10 horas)
  - Interfaz inteligente (boton derecho del raton) (10 horas)
  - Retratos de unidades e interaccion con estos (15 horas)
  - Movimiento suave de camara (5 horas)
- Implementación de un mapa reducido en la ventana de comandos para acceder rápidamente a las distintas zonas de éste. (3 horas)

6.2.3 Logica

1. Completar el protocolo de interacción con los módulos GUI y Gráfico. (10 horas)
2. Completar jerarquía de clases para las unidades y para la IA de las unidades. (20 horas)
3. Completar implementación del comportamiento mediante IA. (20 horas)

6.3 Entrega

Fecha de revisión: Última clase práctica.

Porcentaje a realizar: 20% restante

1. Disponibilidad de varios escenarios. (15 horas)
2. Completar el modelado de las distintas unidades. (40 horas)
3. Finalizacion de menús (5 horas)
4. Depurar errores (10-15 horas con un 75% de fiabilidad, 5-25 horas con un 90%)

6.4 Optativos

1. Leer y guardar partidas
2. Posibilidad de entrar en modo de primera persona (mediante un desplazamiento de cámara)
3. Realización de distintos efectos de luces.
4. Efectos especiales (fuego, tormentas, lasers, sistemas de partículas)
5. Variación de las texturas del mapa según el transcurso del juego (suelo quemado tras un combate, etc.)
6. GUI amigable, mejorada con facilidades al usuario.
7. Posibilidad de jugar en red.
8. Posibilidad de que las unidades se muevan por terreno inclinado
9. Animación de unidades
10. Posibilidad de que las unidades disparen en una direccion distinta de la que caminan (que los tanques giren la torreta para disparar)
11. Implementación de IA de grupo sencilla que asigne prioridades a zonas y de ordendes sencillas a las unidades (moverse a dichas zonas).
12. Selección múltiple de unidades mediante un cuadro de selección.
13. Creación de grupos de unidades (Ctrl+[0..9])
14. Unidades aéreas (helicóptero) movimiento igual que el resto de unidades, solo que el modelo se situa con la coordenada Y valiendo una cierta cantidad más que el suelo. (Antes de hacer esto sería conveniente hacer el apartado de unidades animadas, ya que resultaría un poco sorprenderte ver volar un helicóptero con las aspas paradas...)
15. Indicar la salud de las unidades en el terreno de alguna forma (barra de vida, o color de la marca de selección)
16. Indicar la salud de las unidades en el retrato de selección.
17. Mostrar progreso de tarea de edificio en el area de retratos (construcción, investigación, etc) una barra de progreso.
18. Menu dentro de juego (pausa, opciones de video, juego, etc)
19. Menu de opciones de juego, video y/o sonido. (Es decir, posibilidad de configurar ciertos aspectos del juego, como la calidad de las texturas o la velocidad de scroll)
20. Secuencia de muerte de unidades (explosiones para edificios y vehiculos, elevación del alma para personas...) algo en lugar de hacer desaparecer directamente la unidad.
21. Secuencia de construcción de edificios (hacerlo opaco gradualmente, o hacerlo crecer en el eje Y, o truncarlo en el eje Y cada vez a mas altura...) algun efecto en lugar de hacer aparecer al edificio de repente en cuanto llega el constructor. Durante este tiempo el constructor permanecera junto al edificio (construyendolo claro, no admirando el bonito efecto que hemos hecho..., asi que disponer de animaciones para las unidades ayudaría a distinguir este hecho)
22. Búsqueda de camino para las unidades.
23. Cambio del estado de respuesta de la unidad (estado ofensivo, defensivo, mantener terreno, o pasivo) Si no se hace este optativo las unidades tendrán un unido diagrama de actuación que correspondera al del estado ofensivo (perseguir cualquier unidad que vean hasta donde sea)
24. Depurar en mayor medida los errores de modo que incluso sea posible jugar mas de una partida sin salir del juego.