GoM: Simulador de batallas fantásticas

Alumno: Aarón Bueno Villares
Director: Manuel Palomo Duarte

26 de abril de 2010

Resumen

El siguiente documento se presenta a modo de resumen que complementa la memoria del mismo Proyecto Fin de Carrera, entregado el dia de la presentación de este proyecto. El proyecto que nos ocupa es la creación de un juego de guerra de táctica militar de corte fantástico-medieval, primer juego que sirve como alternativa digital y gratuita de los populares juegos de miniaturas de la misma corte.

Índice

1. Introducción

Los juegos de guerra fantásticos de táctica por turnos son muy demandados dentro del ámbito de los juegos de mesa. Una prueba de ello es la popularidad de la que goza la empresa *Games Workshop*, una empresa que se dedica, entre otras muchas cosas, a mantener dos juegos muy populares y famosos de táctica militar por turnos, *Warhammer Fantasy Battles*, de corte fantástico, y *Warhammer 40k*, de corte futurista.

Debido al éxito entre los jóvenes y adolescentes, cada vez de corta edad, de estos juegos, la empresa tuvo que adaptarse a sus nuevos clientes, y el juego, a gusto de otros públicos, se hizo algo infantil.

También los precios se incrementaron debido a que ahora, los compradores son madres en vez de adolescentes, que tienen menor disponibilidad económica. Este hecho sobre todo fue el incentivo de que gran cantidad de jugadores mas adultos dejaran el hobby.

El siguiente paso obvio fue buscar una alternativa digital a este paradigma de juego. Tal búsqueda resultó una quimera ya que no existía tal alternativa en la oferta de videojuegos. Todos los juegos por turnos que hay son juegos de guerra de corte moderno, es decir, cuyo componente principal es la artillería y las formaciones abiertas con mucha movilidad, y no del elegante paradigma de guerra clásico que existía antes de la popularización de la polvora.

GoM pretende ofrecer (por primera vez en la historia, por cierto) una alternativa digital a este paradigma de juego con tanta popularidad, con el incentivo de ser libre, para incentivar su mas rápida expansión, además de ser gratuito.

GoM, al igual que *Warhammer Fantasy* y otros tantos juegos de táctica (no digitales) que han existido en la historia de los juegos de guerra, está basado en un reglamento que especifica exáctamente como resolverá *GoM* cada una de las acciones, y en concreto, qué acciones son las permitidas.

Los juegos tácticos por turnos tienen el aliciente de tener un mayor control y un conocimiento mas profundo sobre sus tropas, y, además, como también cuentan con el factor suerte (muchas de las decisiones se modelan mediante números aleatorios, como en el resultado de los combates), añaden mas tensión al juego.

2. Calendario

El desarrollo del sistema ha seguido modelo de desarrollo iterativo incremental, donde a cada iteración se completaba e implementaba una nueva faceta de la aplicación, hasta conseguir un reglamento implementado completo, rígido y suficiente.

En la figura 1 podemos ver las iteraciones y/o ciclos en las que se ha divido el desarrollo del sistema:

- Definición inicial (3.1)
- Arquitectura general del sistema (3.2)
- Fase de movimiento (3.3)
- Fase de combate (3.4)
- Interfaz gráfica (3.5)
- Fase de disparos y magia (3.6)
- Memoria (3.7)

3. Iteraciones de desarrollo

Se dedicará una sección para cada una de las iteraciones de desarrollo del proyecto.

3.1. Definición inicial

Esta iteración contiene las reflexiones iniciales del proyecto, así como la obtención de documentación sobre el contexto de *GoM* en la diversidad y oferta de videojuegos, la confección de un reglamento inicial como punto de partida para el desarrollo consecuente y la toma de requisitos generales del mismo: interfaz gráfica 2D, librerias, propiedades del sistema, etc.

3.2. Arquitectura general del sistema

Una vez confeccionado un reglamento, se generó una arquitectura general del sistema que sirviera como punto de partida en su implementación.

Se identificó una capa de dominio llamado gestor de reglas, que implementaría al reglamento y fuera su motor, y una capa de presentación que fuera el gestor de interfaz, que se intercomunicaría con el gestor de reglas para pedirle la información que debe mostrar al usuario.

El gestor de reglas dispone de una clase llamada estado que guarda el estado actual de la partida. El usuario realiza sus acciones enviándo sus ordenes al gestor de interfaz, éste, envía la petición al gestor de reglas. Toda nueva acción provocará un cambio de estado en el sistema; el gestor de reglas, por tanto, con la acción recibida cambiará el estado interno del juego y lo devolverá al gestor de interfaz para que éste lo imprima. El diagrama de clases se muestra en la figura 3.2.
A su vez, el gestor de reglas se descompone en otros gestores mas concretos que se encargan de las acciones específicas de las distintas fases de un turno de juego, como muestra la figura 3.2.

3.3. Fase de movimiento

Ésta es quizás la fase mas crucial de la capa de dominio. La fase de movimiento, como clase del sistema, solo se encarga de organizar y discriminar las acciones disponibles en cada momento dentro de la fase de movimiento, y podemos decir que es una clase sencilla, pero el diseño e implementación de esta clase trajo consigo el diseño e implementación del gestor de escenarios, una clase auxiliar que se encarga de todo el cálculo numérico del juego. Se llama gestor de escenario porque todos estos cálculos vienen referidos a la relación de una unidad con el resto de unidades dentro de una batalla.

Por lo que este ciclo de desarrollo se reduce o es sinónima del desarrollo de este gestor de escenario (3.3). El gestor de interfaz también interactúa con esta clase ya que necesita cierta información para ofrecer al usuario la forma de responder a sus peticiones, por ejemplo, para que un usuario mueva una unidad, necesita saber cual es su desplazamiento máximo. El gestor de interfaz pide esta información al gestor de escenario, hace los cálculos oportunos, y devuelve el dato. El gestor de interfaz a continuación imprime en el escenario un área transparente que indica la zona disponible de movimiento.

3.4. Fase de combate

En esta iteración se analizó, diseñó e implementó ampliamente la fase de combate y todas las acciones involucradas en él. Se hicieron bastantes cambios en el reglamento y luego éstos fueron implementados.

Estructuralmente esta fase es sencilla, aunque posee muchos cálculos discretos cuya mayor dificultad es el propio caos de información que hace falta manejar para gestionar todos y

cada uno de los combates (flancos, unidades combatientes en cada combate, ataques sin asignar, etc).

3.5. Interfaz gráfica

Antes de esta iteración, lo que se tenía en la capa de presentación era sencillamente una interfaz gráfica sencilla que solo servía para mostrar las unidades y poder hacer pruebas y ver los resultados de las implementaciones anteriores.

En esta iteración por fin se resolvió implementar la interfaz diseñada, es decir, el menú y una presentación de la batalla mas estéticos y completo, implementando también al gestor de ejércitos, que permite manejar una serie de ejércitos, que pueden ser creados y modificados por los usuarios, que quedarán grabados en el sistema y podrán ser usados en las partidas.

El gestor de interfaz se diseñó considerando a un gestor de iconos como entidad separada del resto, que se encargaba de mostrar todos los iconos del juego según las acciones disponibles en cada momento de la batalla, imprimiéndolos en el orden y posición adecuada.

A su vez, se usó un gestor de textos (cuya clase se llamó ListaTexto), se permitía manejar de forma abstracta una lista de items textuales a los que poder, de forma sencilla y cómoda, acceder, eliminar, modificar, marcar, etc.

En la figura 3.5 se muestra el diagrama de clases usado para implementar toda la funcionalidad ofrecida por el gestor de interfaz.

3.6. Fases de disparo y magia

En esta iteración se incluyó en el sistema el gestor de disparo mostrado en la figura 3.2 y el sistema de magia que, por su sencillez y simplicidad, se añadió los correspondientes cálculos de control en los puntos adecuados de los otros gestores.

3.7. Memoria

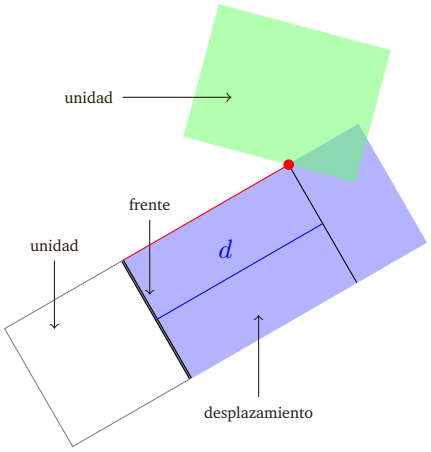
En esta última iteración se realizó la memoria del proyecto.

4. Principales problemas de implementación

Se mostrará algunos de los problemas de implementación numéricos a los que tuve que enfrentarme en la confección del gestor de escenarios.

4.1. Desplazamiento máximo

El problema del desplazamiento máximo es el hecho de saber qué distancia máxima tiene disponible una unidad para desplazarse al frente. Esta capacidad tiene como cota máxima el valor del atributo de movimiento base de la unidad, y se va reduciendo a medida que se encuentran mas unidades por el camino.



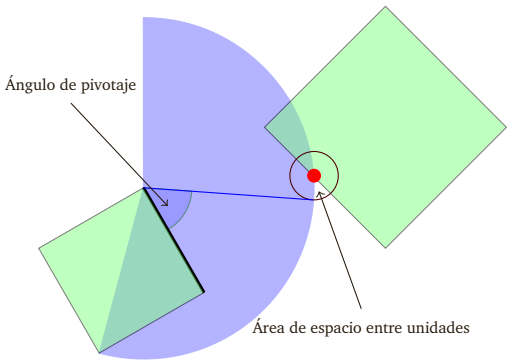
La forma de resolver este problema, como ilustra la figura de la izquierda, consiste en explorar todas las unidades presentes en el campo de batalla y, para cada una de ellas, buscar todas las posibles intersecciones (de aristas y vértices) con el rectángulo de movimiento base, quedándonos con la distancia mínima de entre todas las distancias a estos puntos de intersección.

Para no violar la regla del espacio entre unidades incluida en el reglamento, inicialmente se engrosa a la unidad esta misma cantidad, y así este espacio de respeto queda incluido de forma intrínseca a la búsqueda.

4.2. Pivotaje máximo

De forma análoga al problema anterior, este problema consiste en saber cual es la capacidad de pivotaje máxima de una unidad, en relación al resto de unidades presentes. Pivotar es el acto de que una unidad cambie su encaración como resultado de mover la unidad mantiendo una esquina de la formación como eje.

Para resolver este cálculo, se toma una semicircunferencia cuyo arco mide la capacidad de movimiento base de la unidad. Luego, para cada unidad del campo de batalla, y al igual que en el caso anterior, se toman los puntos de intersección con dicha semicircunferencia.



Como hemos de respetar la regla de la seperación de unidades, para cada punto de intersección tomamos una circunferencia cuyo radio es dicho espacio de separación. Como ilustra la figura, estos puntos obtenidos tras buscar sus rectas tangentes desde el origen de la semicircunferencia original serán los tomados en consideración para ir acotando el ángulo de pivotaje máximo de la unidad.

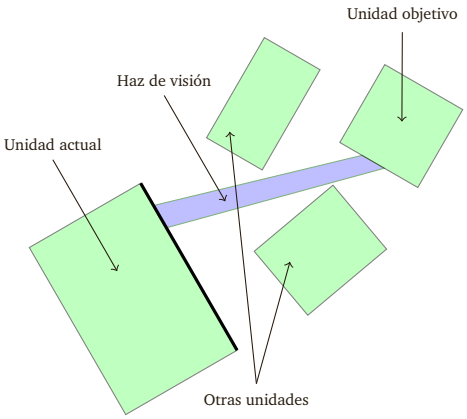
4.3. Visión de una unidad

Caracterizar la visión de una unidad es establecer cuando una unidad ve a otra. Esto depende de la cantidad de objetos que hayan en el camino. A mayor cantidad de objetos, menor es la probabilidad de que una unidad vea a la otra.

Lo que se busca entonces es, para cada pareja de unidades, construir el número máximo de rectángulos posibles, desplazados en cantidades discretas, que se proyecten desde una unidad a la otra.

Para cada rectángulo, si existe al menos una unidad que tenga intersección con dicho rectángulo o haz de visión, el rectángulo será desechado. Si sobrevive al menos un rectángulo, podremos admitir que la primera unidad ve a la segunda.

Dicho de otro modo, en cuanto encontremos el primer rectángulo completamente limpio finalizamos la búsqueda y admitimos que existe visibilidad entre las unidades.

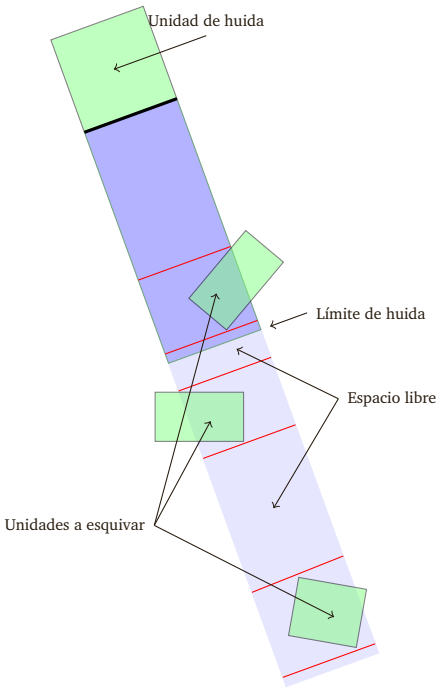


4.4. Movimiento de huida

Cuando una unidad huye, se genera un valor aleatorio entre la mitad y el triple de la capacidad de movimiento de la unidad que va a huir. Si la posición final de la unidad, según este nuevo valor, está ocupado, la posición final de la unidad que huye será el siguiente espacio libre encontrado.

Para ello, en cada unidad presente en el camino, trazamos un par de rectas discriminantes para cada unidad, una justo delante suya, que llamaremos proximal, y otra justo detrás suya, que llamaremos recta distal. Son las que hemos puesto de color rojo en la figura.

Para cada unidad, obtenemos el espacio que existe entre la recta distal de la primera unidad y la recta proximal de la siguiente. Si este espacio es mayor a la profundidad de la unidad, significará que la unidad cabe en dicho espacio, y la primera posición encontrada que cumpla con estas propiedades será la posición elegida final para colocar a la unidad que está huyendo.



5. Pruebas

Hemos distinguido dos fases de pruebas, las pruebas alfa, que son las efectuadas durante el proceso de desarrollo, y las pruebas betas, realizadas al finalizar este por mí y por otros compañeros.

En las pruebas alfa, se pueden distinguir entre pruebas unitarias y pruebas de integración. En las primeras, se diseñaban según criterios de caja blanca y de cobertura de sentencias, esto es, probando la implementación de las funciones vistas desde dentro, y procurando explorar todas las ramas de ejecución posibles. Estas pruebas fueron las usadas principalmente en la capa de dominio del juego. Por el contrario, las pruebas de integración fueron

las usadas en la capa de presentación, pues es aquí donde cobra sentido el hecho de que todas las clases y todo el juego en general actúe de forma conjunta correctamente.

En las pruebas beta se detectaron fallos solamente muy concretos y en ocasiones muy excepcionales del juego, pues la mayoría de los errores fueron satisfactoriamente encontrados con las pruebas unitarias.

6. Conclusiones

En la realización del proyecto he aprendido lo importante que es el buen orden y el uso de la metodología en la creación de software. La creación del software es muy compleja y hacen falta ciertas directrices generales, que por lo general son creadas desde la experiencia de terceras personas. Aunque desde el comienzo hice un análisis y diseño del software, en muchas ocasiones he hechado de menos no haberme detenido aún mas tiempo en el estudio abstracto del problema antes de ponerme directamente a implementar.

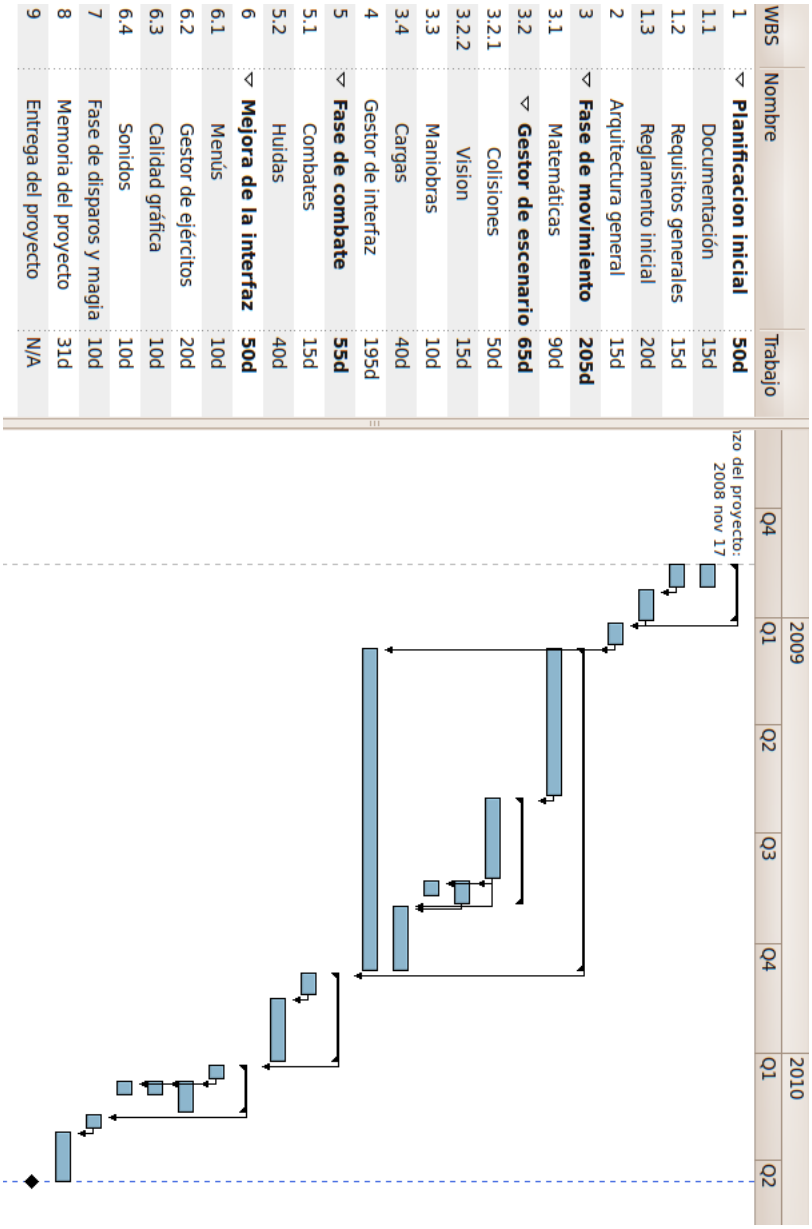


Figura 1: Diagrama de Gantt del calendario del proyecto

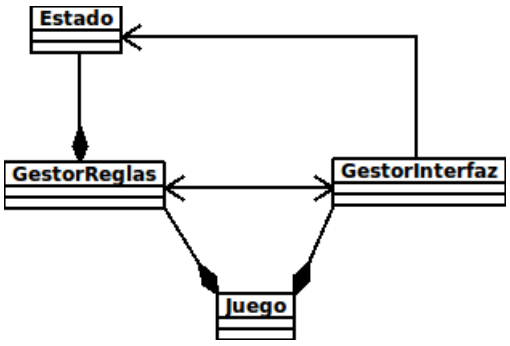


Figura 2: Arquitectura general del sistema

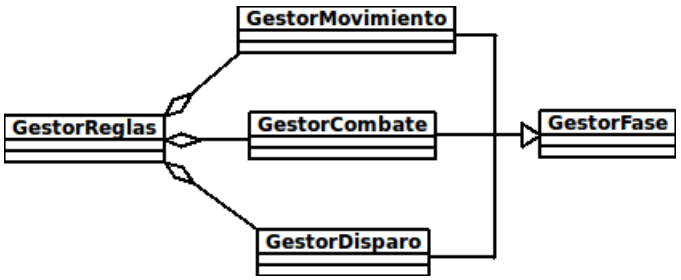


Figura 3: Gestor de reglas

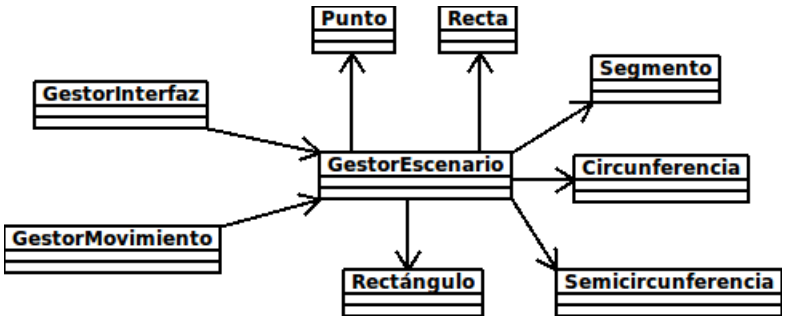


Figura 4: Gestor de escenario

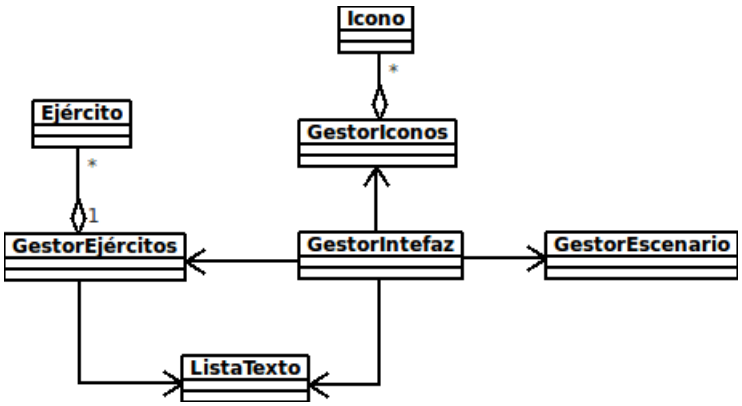


Figura 5: Gestor de intefaz

Referencias

[1] C++ reference wiki. http://www.cppreference.com/wiki/.

[2] Sdl simple directmedia layer official page. http://www.libsdl.org/index.php.

[3] *Warhammer: El juego de batallas fantásticas, reglamento*. Games Workshop.

[4] Antonio Gardia Alba. Wiki-tutorial de libsdl para la programación de videojuegos.

[5] Andrew M. St. Laurent. *Understanding Open Source & Free Software Licensing*. O'Reilly, 2004.

[6] Roger S. Pressman. *Ingeniería del Software: Un enfoque práctico - Quinta Edición*. Mc Graw Hill, 2002.

[7] Eric S. Raymond. *The cathedral and the bazaar*. O'Reilly, 2001.

[8] Richard M. Stallman. *Software libre para una sociedad libre*. Traficantes de sueños: Mapas, 2004.

[9] Frank Mittelbach y Michel Goossens. *The LaTeX Companion*. Addison-Wesley, 2004.