# Diseñando la I.A. de un juego tipo Tycoon

Autor: Sergio Hidalgo
serhid@wired-weasel.com

## Introducción

En este tutorial voy a tratar de explicar cómo diseñamos el sistema de inteligencia artificial para los personajes de nuestro proyecto de videojuego: Theme Faculty. Aunque todo lo que comento es específico para nuestro proyecto en concreto, si que creo que se puede extender a otros juegos parecidos. O si no, al menos puede que sirva para dar alguna idea.

Para orientarnos un poco, el juego es bastante parecido al Theme Hospital, pero ambientado en una universidad en lugar de un hospital. Vamos a tener varias salas o "dependencias", como cafetería, aulas, despachos, baños, etc... En esas salas, y también en los pasillos, tendremos diversos objetos (sillas, pizarras, fuentes, máquinas de bebidas...).

Y por último tenemos una serie de personajes que interactúan entre sí, y con los objetos y dependencias. En lugar de médicos, tendremos profesores, y en lugar de pacientes, alumnos. También tendremos bedeles que se encargan de limpiar y tenerlo todo al día.


**NOTA:** Todo el diseño de este sistema (y su implementación), fue el trabajo de un grupo de varias personas, y no únicamente mio. Quiero dejar claro que si lo explico aquí, es tan solo porque creo que puede ser interesante para alguien más.

**Personajes**

El control del jugador sobre los personajes es bastante limitado. En el caso de los empleados (profesores y bedeles), se pueden contratar y despedir, cambiar su sueldo, y repartir el tiempo que dedicarán a cada una de sus tareas. En el caso de los alumnos, no se tiene ningún control en absoluto.

Antes de seguir, vamos a hacer un repaso a las acciones que queremos que pueda tener cada personaje, y el funcionamiento del sistema en general:

Profesores:

Los profesores reparten su tiempo en tres actividades: Dar clase (a los alumnos), trabajar en sus despachos, y asistir a conferencias.

Cuando un profesor decide dar una clase, avisa a todos los alumnos matriculados de esa carrera, y acude al aula, donde interactúa con un objeto (pizarra, proyector, o similar) durante el tiempo que dura la clase.

Trabajar en el despacho o asistir a conferencias es más sencillo, simplemente va a la dependencia (despacho o sala de conferencias), e interactua con el objeto correspondiente (mesa de despacho, o silla).

Aparte de eso, los profesores también van a la cafetería cuando están cansados, o a los baños cuando sube su nivel de orina.

Alumnos:

Cuando un alumno entra en la universidad, antes que nada, se matricula de una carrera determinada. Después, pasa el tiempo vagueando por los pasillos, asistiendo a algunas clases, en la cafetería, biblioteca, etc... Pasado un tiempo límite, el alumno se examina, y dependiendo de cuanto haya aprendido aprueba o suspende. En cualquier caso, después del examen se va de la universidad.

Para ir a una clase, el alumno primero recibe una señal de ir a clase de parte de un profesor. Si está matriculado, decide si asistir o no en base a un factor aleatorio (y a su estado anímico).

Cuando se acerca el momento del examen, los alumnos también pueden decidir ir un tiempo a la biblioteca y estudiar, aumentando sus probabilidades de aprobar.

Aparte de eso, cuando un alumno está vagueando por un pasillo, puede interactuar con los objetos cercanos, como fuentes o máquinas de bebidas.

Y por último, el sistema de ir a la cafetería o al baño es similar al de los profesores.

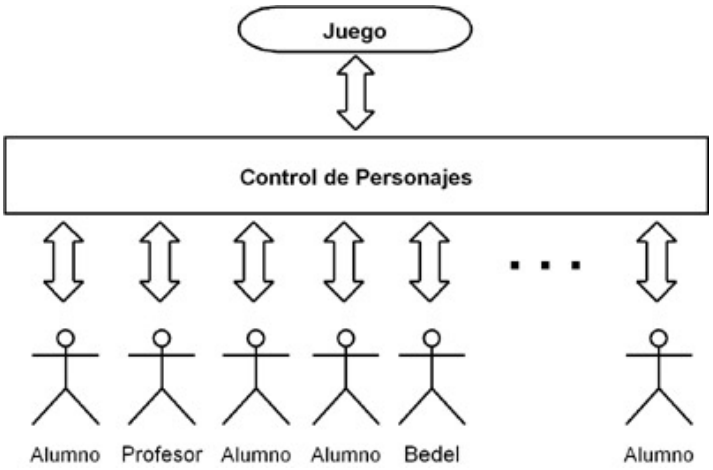
Bedeles:

Los bedeles son el tipo de personaje más sencillo, y no voy a entrar en demasiada profundidad con ellos. Simplemente recorren el mapa limpiando los objetos "basura" que encuentran, y arreglando los objetos dañados.

**Arquitectura**

Vamos a considerar que cada personaje es un ente "autónomo", de forma que tome sus propias decisiones. Como cada personaje es independiente de los demás, y del resto del juego, esto simplifica bastante el diseño.

Para comunicar los personajes entre sí, y con el resto del juego, tendremos una clase "Control de Personajes" que actuará como interfaz.

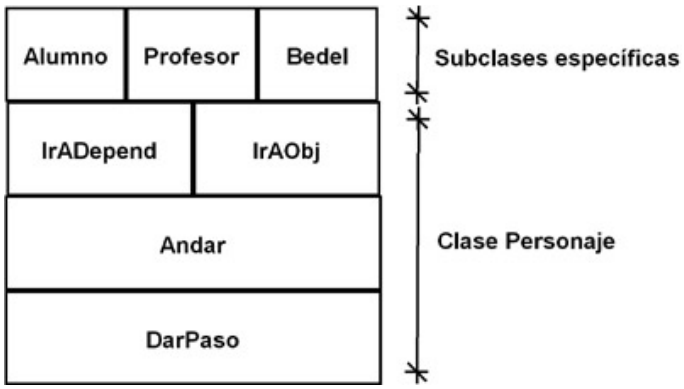



Cada personaje se pasa mensajes únicamente con esta clase de control. Asi que por ejemplo, cuando un profesor decide impartir una clase, el proceso es:

- El profesor envía un mensaje a Control indicando qué clase va a impartir, y en qué dependencia.
- El Control transmite el mensaje a todos los alumnos matriculados de esa carrera.
- Cada alumno decide internamente si asistir o no a clase. Y los que asisten, solicitan al Control de Personajes que les calcule el camino desde su posición hasta la dependencia.
- El Control tramita las peticiones, y se las pasa a la clase del Juego que se encarga de ello. Según se calculan los caminos, se los pasa a cada uno de los alumnos.
- etc...

Además de esta clase de Control, tendremos otras 4 clases para los personajes: Una clase Personaje, que implementa los comportamientos comunes, y otras 3 (Alumno, Profesor, y Bedel), cada una con los atributos y comportamiento específicos de cada tipo de personaje.

Para entenderlo mejor, es bastante útil verlo como un sistema construido en "capas". Cada capa usa las funciones de la capa inferior, y añade comportamientos más complejos:




- **DarPaso:** La capa más baja implementa el comportamiento básico de moverse de una casilla a otra adyacente, a lo que llamamos "dar un paso". Esto depende mucho del tipo de mapa que se use (en nuestro caso isométrico, pero no tiene porque ser así), y no vamos a entrar en detalles por ser bastante simple.

- **Andar:** Usando la capacidad de dar pasos de la capa inferior, construimos un sistema capaz de recorrer caminos largos de un punto del mapa a otro. A esto es a lo que vamos a llamar "Andar".

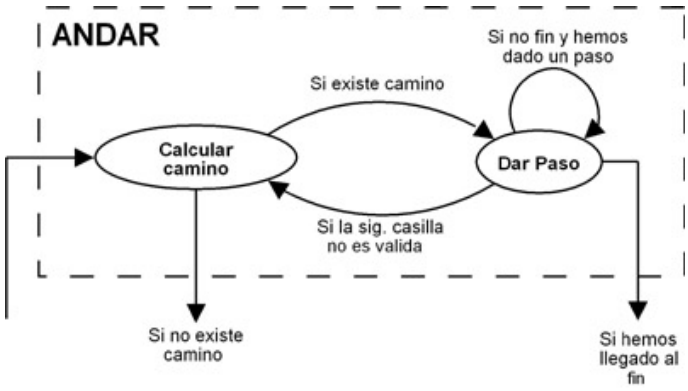
- **IrADep, IrAObj:** Y usando la capacidad de andar, construimos dos sistemas de comportamiento que nos permiten ir a un objeto e interactuar con él (IrAObj), o ir a una dependencia, interactuar con un objeto en su interior, y abandonarla (IrADep).

  Prácticamente todas las acciones de los personajes se pueden clasificar en uno de estos dos patrones.

- **Alumno, Profesor, y Bedel:** Y por último, cada clase específica implementa su propio comportamiento, usando los dos modelos anteriores de IrADep e IrAObj.

Y para ver el funcionamiento de cada una de estas capas lo haremos con unos diagramas de estado, que son bastante más intuitivos que los ladrillos de texto : )

**Andar**

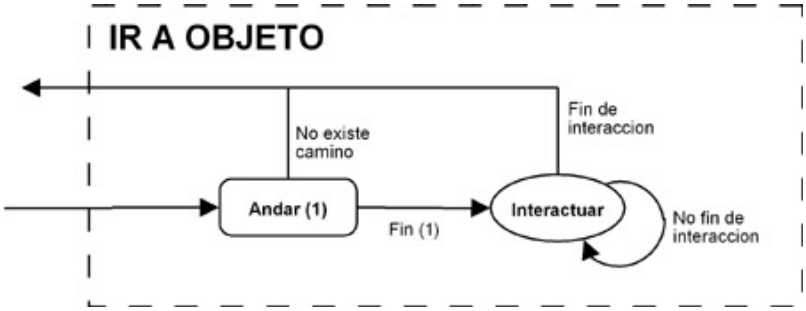



Calcular Camino espera que Control de Personajes envíe el camino calculado entre los dos puntos que queremos recorrer. Cuando esa respuesta llega, si existe un camino válido, se entra en un ciclo de dar pasos para ir recorriéndolo hasta que llegamos a la casilla final.

En el caso de que alguno de los pasos no se pueda dar (por haberse bloqueado el camino, por ejemplo, porque el jugador ha colocado un objeto entre medias) se intenta recalcular de nuevo.

**NOTA:** Como decía antes, el encargado de calcular el camino es Control de Personajes, que a su vez le pasará la responsabilidad a otra clase del Juego. Aquí podemos aprovechar para implementar el algoritmo de Pathfinding que más nos guste, o más optimizado nos parezca.

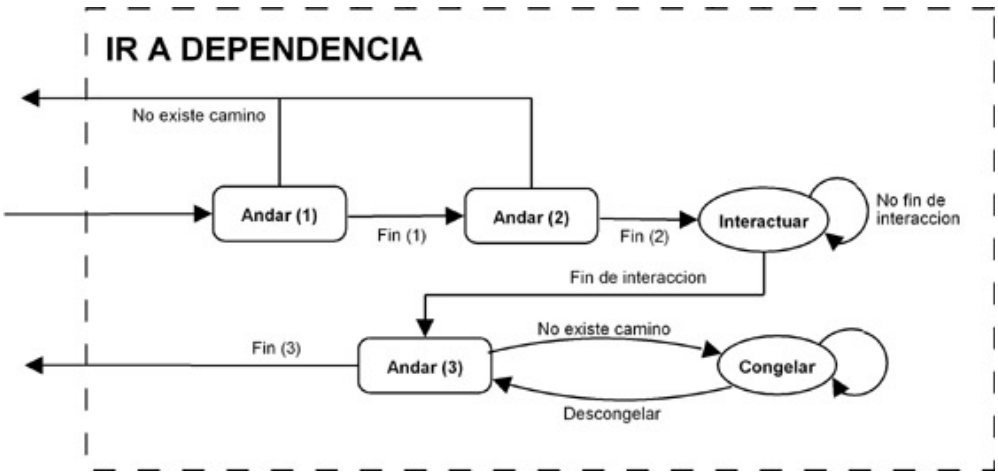
**Ir a Objeto**




Para interactuar con un objeto, en primer lugar entraremos en el estado "Andar" hasta llegar a él, y después entramos en un estado de interacción. En este estado, se reproducen las animaciones correspondientes tanto del objeto como del personaje, y se modifican los atributos dependiendo del tipo de objeto.

Cuando la interacción termina, será el objeto el que enviará una señal para indicarlo**.**

**Ir a Dependencia**

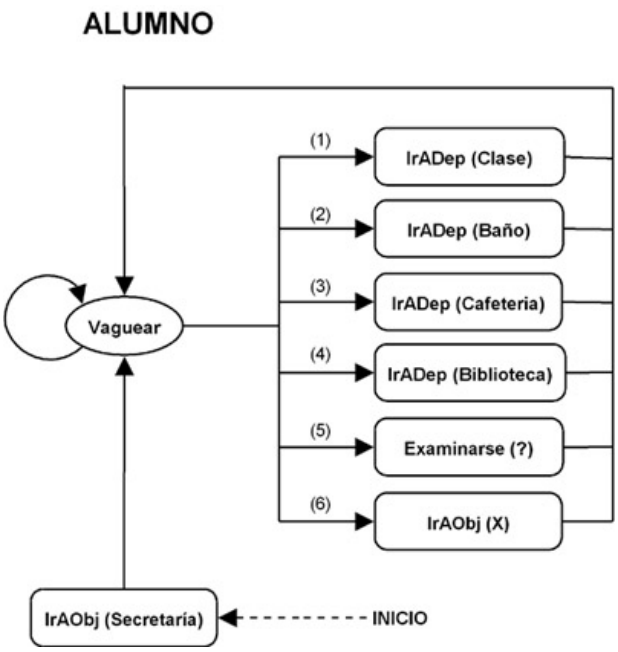



La interacción con las dependencias es algo más compleja. En primer lugar, recorreremos el camino hasta llegar a la puerta de la misma (Andar 1). Ahí se solicitará a la dependencia que nos asigne un objeto para interactuar con él. Una vez lo tengamos, andamos hasta él (Andar 2) e interactuamos de forma similar al caso anterior.

Después calculamos el camino de salida, y abandonamos la dependencia (Andar 3). Si el camino está bloqueado, el personaje se "congela", e intenta recalcular el camino cada cierto tiempo para ver si ya ha terminado el bloqueo.

El congelar el personaje nos evita que se esté continuamente intentando recalcular el camino cada fotograma, con el impacto en el rendimiento que tendría eso. De esta forma, el personaje solo lo intenta una vez cada varios segundos. Podemos aprovechar también para mostrar algún tipo de mensaje al usuario e informarle del bloqueo.

**Clase Alumno**



En el comportamiento de los personajes aparece un estado nuevo: Vaguear. En este estado, el personaje se limita a andar sin rumbo por los pasillos. Las transiciones a los demás estados desde vaguear se producen cuando se cumplen ciertas condiciones:

(1) - Recibe llamada de ir a clase + factor de azar

(2) - Su nivel de orina supera un límite

(3) - Su cansancio supera un límite + factor de azar

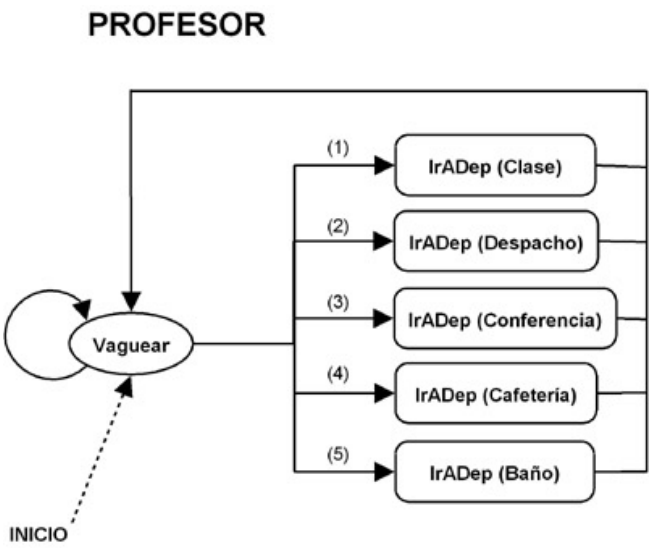
(4) - Se acerca el examen + factor de azar

(5) - Se ha agotado el tiempo de permanencia

(6) - X es un objeto de pasillo cercano + factor de azar

Con ese "factor de azar" añadimos algo de incertidumbre, haciendo que los personajes tengan un comportamiento más impredecible. Esto es en parte un factor aleatorio, y en parte basado en los atributos del personaje en un momento dado (por ejemplo, si el alumno está cansado, hay menos probabilidades de que vaya a clase).

**Clase Profesor**



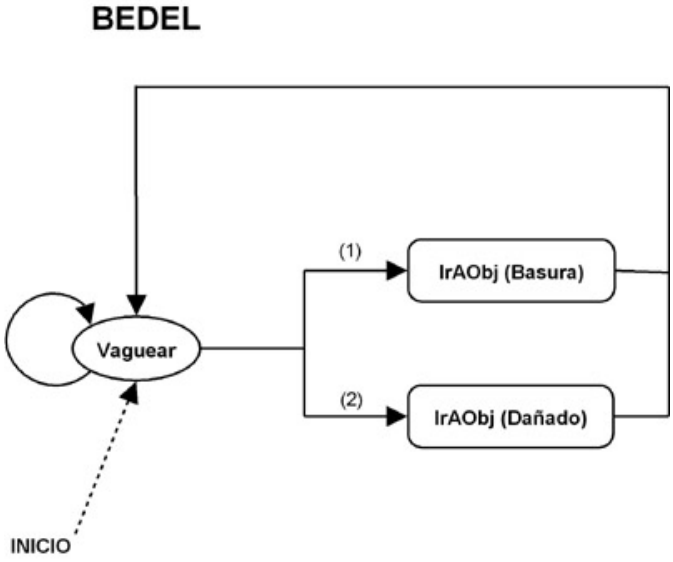

(1) - (2) - (3) - Reparto de tiempos + factor de azar

(4) - Su cansancio supera un límite + factor de azar

(5) - Su nivel de orina supera un límite

En este caso, y en el de los bedeles, aparece el "reparto de tiempo". De esta forma, el jugador puede establecer el porcentaje de su tiempo que el empleado dedica a cada tarea. Por ejemplo, en el caso de un profesor puede hacer que el 50% del tiempo lo dedique a dar clase, el 30% a trabajar en el despacho, y el 20% restante a asistir a conferencias.

**Clase Bedel**




(1) - (2) - Reparto de tiempos

**Señales**

Y por último, lo que queda por explicar es cómo se comunican estas capas que hemos visto antes entre sí.

La comunicación se hace a través de señales. Por ejemplo, cuando acabamos de dar un paso la capa más baja envía una señal que la capa superior (Andar) reconoce, y ésta a su vez envía una señal a "IrADep / IrAObj" cuando el personaje ha terminado de recorrer el camino.

La mayoría de estas señales son internas al personaje (las produce él mismo, y otra capa suya las reconoce). Pero hay también algunas que le llegan desde el exterior (por ejemplo, la que indica el camino calculado que se debe recorrer).

Algunas de estas señales además incluyen información extra en forma de parámetros (de nuevo el mismo ejemplo con la el camino, que deberá dar la información del propio camino mediante algún parámetro).

Esta es una tabla con las distintas señales (al menos, las más importantes):

| Tipo | Parámetro | Generada por: | Descripción |
|---|---|---|---|
| IR_OBJETO | Objeto * | Interna | Indica que se interactúe con el objeto. |
| IR_DEPENDENCIA | Dependencia * | Interna | Indica que se interactúe con la dependencia. |
| CAMINO_CALCULADO | Lista de puntos | Control Personajes | Devuelve el camino solicitado. |
| SIG_PASO | Bool posible | Interna | Indica que se ha dado un paso, y si el siguiente del camino es posible (a través del parámetro). |
| FIN_INTERACTUAR | - | Objeto | Indica cuándo se ha terminado de interactuar con el objeto. |
| DESCONGELAR | - | Interna | Se genera cada cierto tiempo al estar congelado para intentar recalcular el camino. |

**NOTA:** Algunas señales aparecen como internas pero tienen excepciones. Un ejemplo es la de IR_OBJETO, que normalmente es interna. Pero cuando creamos un Alumno nuevo, siempre le vamos a forzar a matricularse. Esta señal se la enviaremos siempre nada más crearlo a través del Control de Personajes.

**Conclusiones**

Está claro que gran parte de lo que he comentado aquí es único para nuestro proyecto (y espero no haberos aburrido demasiado con ello), y cada uno tiene que buscar las soluciones que mejor le funcionen en su caso. Pero creo que una aproximación similar a la que hemos hecho nosotros puede ser bastante útil en muchos casos similares (o no tan similares).

El que los personajes se consideren agentes independientes entre sí simplifica mucho el diseño y la comunicación entre ellos. Además, que en el caso de tener que depurar bugs o ajustar el comportamiento de un personaje, sabemos inmediatamente localizar el problema.

El usar una única clase como interfaz entre los personajes y el resto del juego, y también para comunicarlos entre sí nos permite tanto olvidarnos de los detalles de la implementación del resto del juego cuando trabajamos en los personajes, como olvidarnos de los personajes cuando trabajamos en otras partes del juego.

Pero lo que nos fue más útil al final fue la estructura por capas. Al separar los comportamientos de esta forma, somos capaces de añadir una estructura interna a la IA de los personajes, podemos organizar el código más limpiamente, y reutilizar casi todo el código en los distintos tipos de personajes que teníamos.

Esto también nos permitió el tener a varias personas trabajando al mismo tiempo en la IA de los personajes (en distintas capas), y depurar más rápidamente, al poder probar las capas independientemente unas de otras.

Aunque en el día a día las cosas no fueron tan bonitas, y tuvimos bastantes problemas por la falta de tiempo para implementar esto de la forma que debíamos, si creo que fue un buen diseño, y que sin él posiblemente hubiésemos tenido muchos más problemas de los que tuvimos.

Y esto es todo. Espero que os haya parecido interesante y no os hayais aburrido mucho con los detalles.

Un saludo!