Examen Final de GRAU-IA

(26 de junio de 2018)
Duración: 2 horas 40 minutos

1. (5 puntos) Una empresa del sector de los videojuegos quiere ir mucho más allá en la introducción de la Inteligencia Artificial en sus productos. Para su próximo videojuego quieren que la IA del juego adapte la historia, el ritmo y el tipo de juego al tipo de jugador que lo usa, de forma que tenga más o menos acción, más o menos aventura, más o menos misterio, centrar la historia en el jugador individual o modificarla para que sea un juego de estrategia por equipos, todo ello dentro del mismo juego. Para hacer esta adaptación el sistema dispone del historial de jugador, tanto dentro del juego como en otros juegos de la misma compañia, extrayendo de ahí el perfil de jugador.

   De cada juego del catálogo de esta empresa tiene asociado un título, un año de publicación, la edad recomendada (+0 años, +3 años, +5 años, +8 años, +12 años, +18 años), si es un juego on-line o no, si es un juego multi-jugador y la saga de juegos a la que pertenece (de la que solo se guarda el título y los juegos que la componen). La empresa solo ofrece en estos momentos ocho tipos de juego: arcade, juegos de estrategia en tiempo real (más conocidos por las siglas RTS en inglés), juegos de lucha, de deportes, aventuras gráficas, puzzles, simuladores de mundo y simuladores de vehículos, y de todos ellos se guarda la misma información excepto en el caso de los simuladores de vehículos, en los que se guarda además el conjunto de vehículos que los jugadores pueden pilotar. Cada juego esta compuesto por uno o más niveles, y de cada nivel se guarda su nombre, la dificultad (de 1 a 10), el conjunto de niveles desde los que se puede acceder a ese nivel, y el tiempo máximo para poder superar el nivel (en segundos, si vale 0 significa qe no hay límite de tiempo). Cada nivel está compuesto por uno o más espacios, y de cada uno se guarda su nombre, si es un espacio interior o exterior, el conjunto de espacios desde los que se puede acceder a ese espacio y el conjunto de objetos que estan situados dentro de ese espacio. De cada objeto se guarda su nombre y su precio en créditos (en moneda ficticia, el usuario puede gastar esos créditos en el mismo juego o en otros juegos de la misma compañia). Hay cinco tipos de objetos: las llaves, de las que se guarda además el espacio al que pueden dar acceso; las armas, de las que se guarda nivel de daño que pueden generar por ataque y la energia disponible (en %); los vehículos, que pueden ser aereos, espaciales, marinos, anfibios o terrestres, y de los que se guarda el nombre, el nivel de energía y el número de plazas; las herramientas, de las que se guarda solo el nombre y el precio, y los contenedores, de los que se guarda el conjunto de objetos que tienen dentro.

   La participación de los jugadores en los juegos se guarda por partidas, y de cada partida se guarda la fecha y hora de inicio, la duración (en minutos), los niveles del juego completados por el usuario, los personajes que han participado y los equipos que han participado. En un juego hay uno o varios personajes, que pueden ser personajes jugador (personajes que corresponden a usuarios reales) y personajes no jugador (más conocidos por el acrónimo NPC, personajes que corresponden a jugadores virtuales que crea el juego). De los personajes jugador se guarda el nombre del personaje (texto), el usuario real asociado a este personaje su rol (texto), su salud (en %), los puntos obtenidos (entero), los creditos obtenidos (en la moneda fictícia) y el conjunto de objetos del juego que tiene (ya sea porque los encontró o porque los compró). De los personajes no jugador se guarda el nombre del personaje (texto), su rol (texto), su salud (en %), si es enemigo, y el nivel de inteligencia artificial (de 1 a 10). Tanto los personajes jugador como los no jugador pueden formar un equipo, del que se guarda su nombre y todos sus integrantes. De cada usuario se guarda el nombre de usuario, todas las partidas que ha jugado y el crédito total acumulado (en la moneda fictícia).

   Los personajes pueden realizar acciones, y cada acción tiene asociado un nombre, la fecha y hora en la que se completó la acción, el tiempo de reacción del personaje a la hora de ejecutar la acción (en milisegundos), el conjunto de acciones de las que esta acción depende para poder ejecutarse, el espacio en el que se ha realizado la acción y el conjunto de objetos que requiere la acción para poder hacerla. Los personajes también pueden participar en un diálogo, y de los diálogos se guarda el identificador de diálogo y el conjunto de personajes que ha participado. Todos los juegos (incluso los puzzles de habilidad más simples) tienen una historia asociada, ligada a lo que el jugador va haciendo en el juego. La historia está compuesta por hitos argumentales, es decir puntos importantes dentro de la historia del juego a los que el jugador ha de llegar. Cada hito argumental ocurre en un nivel del juego (un nivel puede tener varios hitos asociados) y tiene asociada una acción objetivo que el jugador ha de conseguir realizar. La información que se guarda para cada hito incluye el tema (una misión individual, una misión grupal, una relación personal, un misterio), si el hito es central (es decir, imprescindible para el juego) o si es opcional (se puede añadir o eliminar sin que el juego pierda sentido), la acción a realizar, la fecha y hora límite para realizar la acción (todo a ceros si no hay límite temporal). Todos los hitos argumentales tienen asociadas tres escenas pre-grabadas de animación 3D: la escena de inicio (un cortometraje donde se narra parte de la historia y le dan pistas al jugador de la acción objetivo que ha de realizar) la escena de objetivo cumplido (se confirma al jugador que ha conseguido el objetivo del hito, y se cuenta un trozo de historia que es consecuencia del éxito) y

la de objetivo no cumplido (un trozo de historia consecuencia del fracaso), de estas escenas solo se guarda el identificador del video y la duración de la escena. Además un hito argumental puede incluir uno o más dialogos (que se insertan en el juego para guiar al usuario en cierta dirección, o darle pistas).

Los servidores de la empresa recopilan toda esta información y la complementan con otros estadísticos generados de forma automática, como el tiempo medio de reacción del jugador (en milisegundos) en las acciones de la partida actual / en el día actual / en la última semana / en el último mes; o el tiempo extra (en segundos) que el usuario dedica a explorar los espacios en la partida actual / en el día actual / en la última semana / en el último mes, para las escenas animadas el porcentaje de video (en media) que ve el usuario en la partida actual / en el día actual / en la última semana / en el último mes (hay usuarios que veran normalmente los videos enteros, otros que se saltarán el video para ir a la acción, pero esto puede variar por dias, según si el usuario tiene tiempo para verlos), etc.

Por lo que nos dicen los expertos se suelen usar varias características para construir un perfil de jugador:

- La **salud** del personaje en la partida actual, que puede ser muy baja ($< 10\%$) , baja ($< 40\%$), normal ($< 80\%$) o elevada ($\geq 80\%$).
- El **tiempo de reacción** medio de las acciones en la partida actual, que puede ser rápido (< 0.4 seg), normal (< 0.9 seg) o lento (≥ 0.9 seg).
- El **tiempo de exploración** medio en las partidas del día actual, que puede ser mínimo (< 5 seg), normal (< 50 seg) o elevado (≥ 50 seg).
- El **tiempo de narración** medio que admite el usuario en las escenas animadas del día actual, que puede ser mínimo ($< 3\%$), bajo ($< 20\%$), normal ($< 80\%$) o elevado ($\geq 80\%$).
- El **nivel de trabajo en equipo**, que está asociado al porcentaje de partidas jugadas e el último mes en las que el usuario ha participado dentro de un equipo, y que puede ser muy bajo ($< 10\%$) , bajo ($< 40\%$), normal ($< 80\%$) o elevado ($\geq 80\%$).
- La **interacción con otros personajes**, que está asociada al número de diálogos en los que el usuario ha participado en el último mes, y que puede ser muy baja (< 5) , baja (< 20), normal (< 30) o elevada (≥ 30).
- La **tipología de juegos** que el usuario ha jugado en el último año, que puede ser variada (un poco de todos los tipos de juego) mayormente de acción (si la mayoría de partidas jugadas son de juegos de tipo arcade o lucha), mayormente de ingenio (si la mayoria de juegos son de tipo puzzle, aventura gráfica o simulador de mundo), o mayormente competitivo (si la mayoría de juegos son de tipo deporte o simulador de vehículo).

A partir de estas características queremos adaptar la partida actual modificando algunos de sus parámetros para ajustar la experiencia de juego a lo que el usuario parece necesitar o preferir:

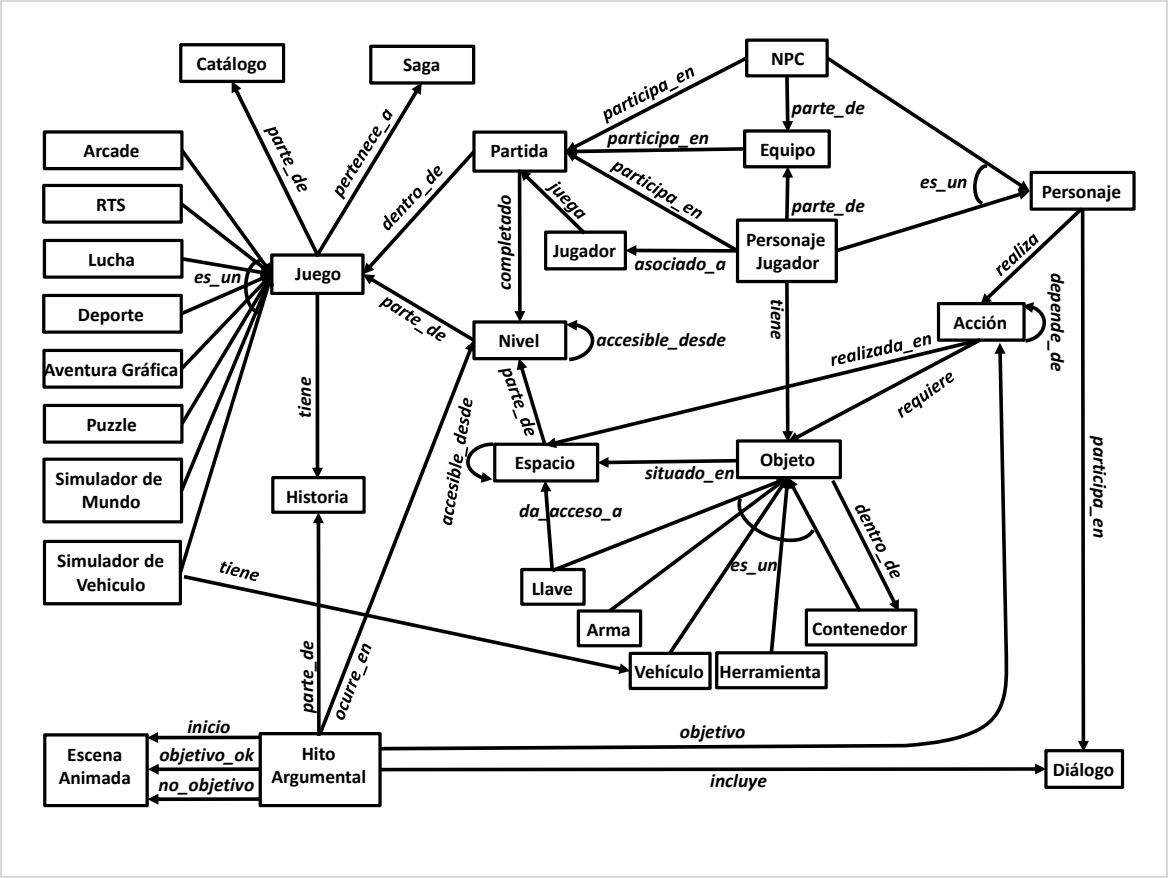
- *Ajustar el número de enemigos*: si la salud del personaje es alta o el tiempo de reacción es rápido y la tipología de juegos es mayormente de acción o mayormente competitivo entonces hay que incrementar el número de NPCs enemigos; si la salud es normal y el tiempo de reacción es normal o alto y la tipología de juegos es mayormente de acción entonces hay que incrementar el número de NPCs enemigos; si la salud es muy baja o si el tiempo de reacción es lento entonces hay que reducir el número de NPCs enemigos; en los otro casos no modificar los enemigos.
- *Ajustar el nivel de la IA de los enemigos*: si la salud del personaje es alta o el tiempo de reacción es rápido y la tipología de juegos es mayormente de aventuras o mayormente competitivo entonces incrementar el nivel de la IA de los NPCs enemigos; si la salud es normal y el tiempo de reacción es normal o alto y la tipología de juegos es mayormente competitivo entonces hay que incrementar el nivel de la IA de los NPCs enemigos; si la salud es muy baja o si el tiempo de reacción es lento entonces hay que reducir el nivel de la IA de los NPCs enemigos; en los otro casos no modificar la IA de los enemigos.
- *Ajustar misiones de equipo*: si el nivel de trabajo en equipo es bajo y el tiempo de narración es normal o elevado entonces introducir hitos argumentales con tema de misión grupal; si el nivel de trabajo en equipo es elevado y el tiempo de narración es elevado entonces introducir hitos argumentales con tema misión individual; si el tiempo de narración es mínimo entonces eliminar todos los hitos argumentales con tema misión individual o misión grupal que no sean centrales; en el resto de casos no variar.
- *Ajustar misterios*: si la tipología de juegos es mayormente de ingenio y el tiempo de narración es elevado o el tiempo de exploración es elevado entonces introducir hitos argumentales con tema misterio; si el tiempo de narración es bajo o el tiempo de exploración es bajo entonces reducir hitos argumentales con tema misterio; si el tiempo de narración es mínimo entonces eliminar todos los hitos argumentales con tema misterio que no sean centrales; en el resto de casos no variar.
- *Ajustar relaciones*: si el nivel de interacción con otros personajes es bajo o muy bajo y el tiempo de narración es normal o elevado y el tiempo de exploración es normal o elevado entonces introducir hitos argumentales con tema relación personajes; si el nivel de interacción con otros personajes es normal y el tiempo de narración es bajo entonces reducir hitos argumentales con tema relación personajes; si

el tiempo de narración es mínimo entonces eliminar todos los hitos argumentales con tema relación personajes que no sean centrales; en el resto de casos no variar.

La salida de este sistema de adaptación ha de sugerir exactamente que modificaciones hay que introducir en el juego (número de enemigos a añadir/eliminar, niveles de IA que aumentar/disminuir, que hitos hay que añadir o eliminar, etc...).

a. (2 puntos) Diseña la ontología del dominio descrito, incluyendo todos los conceptos que aparecen en la descripción e identificando los atributos más relevantes. Lista que conceptos forman parte de los datos de entrada del problema y que conceptos forman parte de la solución. (Nota: tened en cuenta que la ontología puede necesitar modificaciones para adaptarla al apartado siguiente).

En este problema la ontología ha de incorporar, como mínimo, todos los conceptos que forman parte de la entrada del problema: la información que se obtiene directamente de la base de datos de la empresa de videojuegos y la información derivada. Todos los conceptos del diagrama forman parte de la entrada del problema, y algunos de ellos se ven afectados o modificados por la solución concreta final (`NPC` e `Hito Argumental` son los conceptos afectados directamente por la solución, pero es posible que todo concepto que esté directamente relacionado (`Equipo`, `Dialogo`, `Acción`, `Objeto`, `Escena` ...) también se vean afectados (por ejemplo, si hay que añadir más NPC's enemigos eso puede afectar a uno o más equipos, al introducir más personajes puede que se creen nuevos dialogos y nuevas acciones, que requieran espacios y objetos nuevos, etc).



De la ontología resultante cabe remarcar la distinción entre `Jugador` (el usuario o persona del mundo real que juega con los juegos de la compañia) y `PersonajeJugador` (uno de los personajes con los que ek jugador participa dentro del juego). A la hora de perfilar el usuario separaremos la información de la partida actual o bien en la partida actual o en el/los personaje/s que tenga el usuario con información del perfil que cubre varias partidas, que estará dentro del concepto `Jugador`. Tambiñen cabe remarcar que existen varios conceptos que comparten atributos y/o relaciones, y que esto ha sido algo que se ha tenido muy en cuenta a la hora de crear super-clases con las características comunes, o para decidir entre crear subclases o solo añadir un atributo `tipo`. La regla general es que hemos creado subclases cuando una o varias tienen atributos diferentes y/o relaciones diferentes. Ese es el caso de las subclases de `Juego`, `Personaje` y `Objeto`. Siguiendo ese criterio no se crearon subclases

de `Hito Argumental` o de `Vehículo`. También se ha intentado seguir el heurístico del diseño de ontologías que recomienda que ramas hermanas de la taxonomía tengan un nivel de granularidad similar, y por ello, aunque `Vehículo` y `Arma` comparten un atributo de energía, no se ha creado una superclase `ObjetoConEnergía` para no tener una jerarquia de objetos con diferents niveles en las ramas.Otra cosa importante a destacar es que siempre que el enunciado nos dice que un concepto A se compone de B hemos usado la relación taxonómica B `parte_de` A, que modela correctamente como un concepto se compone de sus partes. Y que hemos colocado exactamente tres relaciones entre `Hito Argumental` y `Escena` que establece las tres posibles animaciones asociadas a la escena.

Atributos: a continuación se listan los atributos mínimos para representar la información mencionada explícitamente en el enunciado y la necesaria para el apartado b) del problema.

- `Juego`: título (string), año_publicación (entero), edad_recomendada (enumeración: {+0años, +3años, +5años, +8años, +12años, +18años}), online? (booleano), multijugador (booleano);
- `Saga` título (string);
- `Partida`: fecha_inicio (fecha), hora_inicio (hora), duración (minutos), AJUSTAR_NUM_ENEMIGOS (enumeración: {incrementar, mantener, reducir}), AJUSTAR_NIVEL_IA_ENEMIGOS (enumeración: {incrementar, mantener, reducir}), AJUSTAR_MISIONES_EQUIPO (enumeración: {incrementar_grupal, incrementar_individual, mantener, reducir_grupal, reducir_individual, eliminar_extras}), AJUSTAR_MISTERIOS (enumeración: {incrementar, mantener, reducir, eliminar_extras}), AJUSTAR_RELACIONES (enumeración: {incrementar, mantener, reducir, eliminar_extras});
- `Jugador`: username (string), crédito_total (real), *Tiempo_Exploración* (enumeración: {mínimo, normal, elevado}), *Tiempo_Narración* (enumeración: {mínimo, normal, elevado}), *Nivel_Trabajo_Equipo* (enumeración: {muy bajo, bajo, normal, elevado}) *Interacción_con_Otros_Personajes* (enumeración: {muy baja, baja, normal, elevada}) *Tipología_Juegos* (enumeración: {variada, may_acción, may_ingenio, _competitivo});
- `Personaje`: nombre (string), rol (string), salud (%);
- `NPC`: enemigo? (booleano), nivel_IA (enumeración: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10});
- `PersonajeJugador`: puntos_obtenidos (real), crédito_obtenido (real), *Nivel_Salud* (enumeración: {muy bajo, bajo, normal, elevado}), *Tiempo_Reacción* (enumeración: {rápido, normal, lento});
- `Equipo`: nombre (string);
- `Acción`: nombre (string), fecha_inicio (fecha), hora_inicio (hora), tiempo_reacción (milisegundos);
- `Diálogo`: id_diálogo (string);
- `Nivel`: nombre (string), dificultad (enumeración: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}), tiempo_máximo (minutos);
- `Espacio`: nombre (string), interior? (booleano);
- `Objeto`: nombre (string), precio_en_créditos (real);
- `Arma`: nivel_daño (real), energia_disponible (%);
- `Vehículo`: tipo (enumeración: {aereos, espaciales, marinos, anfibios, terrestres}), número_plazas (entero), energia_disponible (%);
- `HitoArgumental`: tema (enumeración: {misión_individual, misión_grupal, relación_personal, misterio}), central? (booleano), fecha (fecha_max_objetivo), hora_max_objetivo (hora);
- `Escena`: id_video (string), duración (segundos);

Como se puede ver, hemos decidido también representar las características abstractas del problema, las de la solución abstracta y la solución concreta (son los que tienen el nombre en *Cursiva*, *CURSIVA* o MAYUSCULAS, respectivamente). De esta manera todos los conceptos que aparecerán en las reglas están soportados por la ontología.

b. (2 puntos) El problema descrito es un problema de análisis. Explica cómo lo resolverías usando clasificación heurística, usando los conceptos de la ontología desarrollada en el apartado anterior. Da al menos 4 ejemplos de reglas para cada una de las fases de esta metodología.

Para resolverlo mediante clasificación heurística debemos identificar en el problema las diferentes fases y elementos de esta metodología. En este caso hay solo una opción posible: la solución que pide el enunciado solo puede ser una solución concreta, ya que es imposible poder calcular en el nivel de solución abstracta exactamente cuantos enemigos hay que añadir al juego, o que HitoArgumental en concreto vamos a añadir o eliminar sin acceder a los datos concretos como el porcentaje exacto de salud del personaje jugador o cuantos enemigos hay ya cerca del Personaje.

Una vez sabemos cuantas fases requiere nuestra solución ya podemos enunciar la solución completa.

El primer elemento son los problemas concretos que hay que tratar. En este dominio los problemas concretos están definidos por toda la información detallada que la compañia tiene del jugador en todos los juegos de su catálogo y que se ha identificado en la ontología del apartado anterior. Tal y como dice el enunciado, supondremos la existencia de funciones (como tiempo_medio_reacción_partida) que o bien calculan bajo demanda cierto estadístico o van a buscar el valor precalculado en alguna base de datos.

El segundo elemento son los problemas abstractos, estos estarán definidos a partir de las siete características que menciona el enunciado (*Nivel_Salud*, *Tiempo_Reacción*, *Tiempo_Exploración*, *Tiempo_Narración*, *Nivel_Trabajo_Equipo*, *Interacción_con_Otros_Personajes* y *Tipología_Juegos*), todas ellas con rangos de valores discretos.

Para conectar los problemas concretos con los abstractos necesitamos definir las reglas de abstracción de datos, por ejemplo:

- si PersonajeJugador.salud $<$ 10% entonces Nivel_Salud = muy bajo;
- si PersonajeJugador.salud $>$ 80% entonces Nivel_Salud = elevado;
- si tiempo_medio_reacción_partida(PersonajeJugador) $<$ 0.4 seg
  entonces Tiempo_Reacción = rápido;
- si tiempo_medio_reacción_partida(PersonajeJugador) $\geq$ 0.9 seg
  entonces Tiempo_Reacción = lento;
- si tiempo_medio_exploración_hoy(Jugador) $<$ 5 seg
  entonces Tiempo_Exploración = mínimo;
- si tiempo_medio_exploración_hoy(Jugador) $\geq$ 50 seg
  entonces Tiempo_Exploración = elevado;
- si tiempo_medio_narración_hoy(Jugador) $<$ 3%
  entonces Tiempo_Narración = rápido;
- si porcentaje_partidas_equipo_jugadas(Jugador) $<$ 10%
  entonces Nivel_Trabajo_Equipo = bajo;
- si num_dialogos_jugador_mes(Jugador) $\geq$ 30
  entonces Interacción_con_Otros_Personajes = elevada;
- si (porcentaje_partidas(Jugador,arcade) + porcentaje_partidas(Jugador,lucha)) $\leq$ 60%
  entonces Tipología_Juegos = may_acción;
- si (porcentaje_partidas(Jugador, SimuladorVehículo)
  + porcentaje_partidas(Jugador, Deporte)) $\leq$ 60%
  entonces Tipología_Juegos = may_competitivo;

El tercer elemento son las soluciones abstractas. En este caso tenemos solo cinco soluciones abstractas: AJUSTAR_NUM_ENEMIGOS, AJUSTAR_NIVEL_IA_ENEMIGOS, AJUSTAR_MISIONES_EQUIPO , AJUSTAR_MISTERIOS y AJUSTAR_RELACIONES.

Para ligar los problemas abstractos con las soluciones abstractas necesitaremos reglas de asociación heurística, como por ejemplo:

- si (Nivel_Salud == alta o Tiempo_Reacción == rápido)
  y Tipología_Juegos==(may_acción o may_competitivo)
  entonces Partida.AJUSTAR_NUM_ENEMIGOS=incrementar;
- si Nivel_Salud == normal y Tiempo_Reacción == normal
  y Tipología_Juegos==may_acción
  entonces Partida.AJUSTAR_NUM_ENEMIGOS=incrementar;
- si Nivel_Salud == muy bajo y Tiempo_Reacción == lento
  entonces Partida.AJUSTAR_NUM_ENEMIGOS=reducir;
- si (Nivel_Salud == alta o Tiempo_Reacción == rápido)
  y Tipología_Juegos==(may_aventuras o may_competitivo)
  entonces Partida.AJUSTAR_NIVEL_IA_ENEMIGOS=incrementar;
- si Nivel_Salud == normal y Tiempo_Reacción == normal
  y Tipología_Juegos==may_competitivo
  entonces Partida.AJUSTAR_NIVEL_IA_ENEMIGOS=incrementar;
- si Nivel_Salud == muy bajo y Tiempo_Reacción == lento
  entonces Partida.AJUSTAR_NIVEL_IA_ENEMIGOS=reducir;
- si Nivel_Trabajo_Equipo == muy bajo y Tiempo_Reacción == (normal o elevado)
  entonces Partida.AJUSTAR_MISIONES_EQUIPO=incrementar_grupal;

- si Tipología_Juegos == may_ingenio y
  (Tiempo_Narración == elevado o Tiempo_Exploración == elevado)
  entonces Partida.AJUSTAR_MISTERIOS=incrementar;
- si Tiempo_Narración == mínimo entonces
  (Partida.AJUSTAR_MISIONES_EQUIPO=eliminar_extras
  y Partida.AJUSTAR_MISTERIOS=eliminar_extras
  y Partida.AJUSTAR_RELACIONES=eliminar_extras)

El cuarto y último elemento son las soluciones concretas. En este caso corresponde al cálculo de los ajustes exactos a aplicar en la partida actual, echando mano de la solución abstracta y de los datos concretos del problema. Supondremos la existencia de funciones que combinan la solución abstracta con datos concretos del problema y aplican directamente cambis a la partida actual. Lo que sigue son algunos ejemplos de reglas:

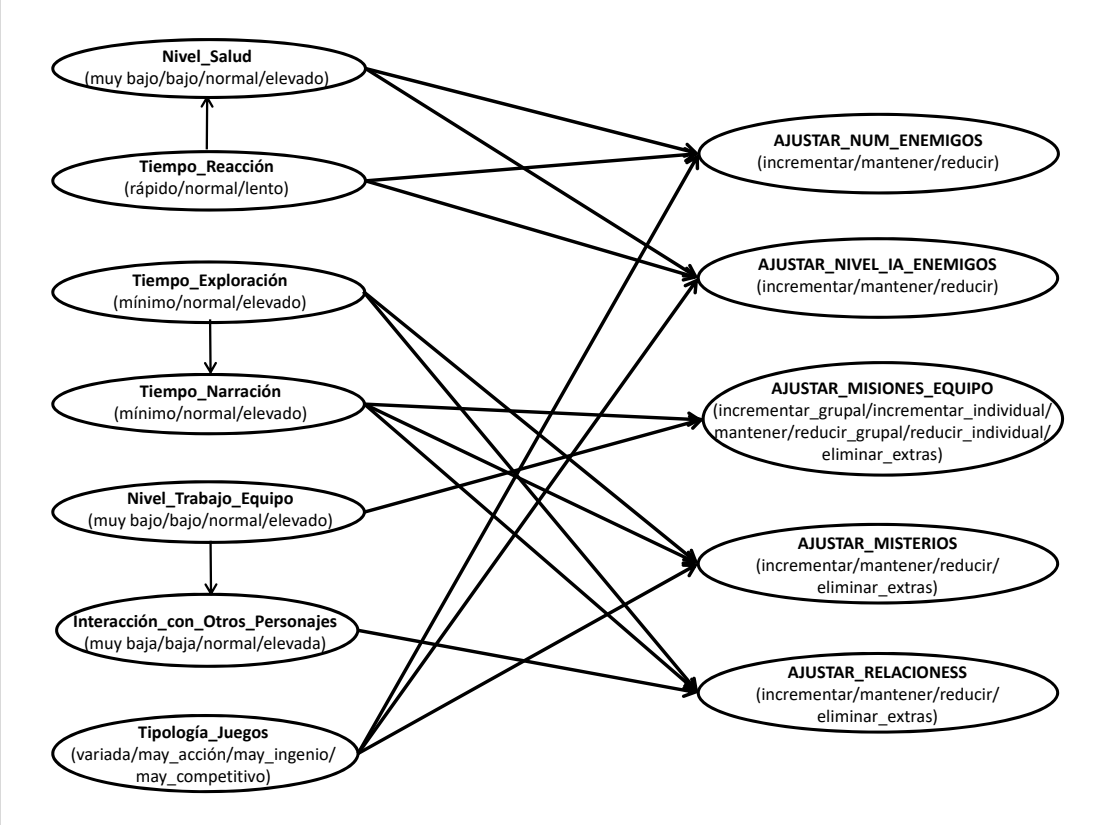
- si Partida.AJUSTAR_NUM_ENEMIGOS==reducir entonces
  reduce_num_Enemigos(PersonajeJugador.salud, PersonajeJugador.tiempo_medio_reacción);
- si Partida.AJUSTAR_NUM_ENEMIGOS==incrementar entonces
  aumenta_num_Enemigos(PersonajeJugador.salud, PersonajeJugador.tiempo_medio_reacción);
- si Partida.AJUSTAR_NIVEL_IA_ENEMIGOS==incrementar entonces
  aumenta_IA_Enemigos(PersonajeJugador.salud, PersonajeJugador.tiempo_medio_reacción);
- si Partida.AJUSTAR_MISIONES_EQUIPO==incrementar_grupal entonces
  añade_Hito_Argumental(PersonajeJugador, Nivel);
- si Partida.AJUSTAR_RELACIONES==eliminar_extras entonces
  elimina_Hitos_Argumentales_extra(PersonajeJugador, Historia);

c. (1 punto) Las características que se usan durante el proceso de adaptación del videojuego no son independientes entre sí. Los tiempos de reacción suelen influir en la salud del personaje (con tiempos de reacción lentos la salud suele empeorar); el tiempo de exploración suele influir en el tiempo de narración (un usuario que dedica mayor tiempo a explorar el entorno suele tener más paciencia o curiosidad por ver los videos enteros, y por ello el tiempo de narración es mayor, mientras que un tiempo de exploración menor suele asociarse a un jugador con menos tiempo disponible o menos curiosidad, y por lo tanto tener un tiempo de narración menor); el nivel de trabajo en equipo influye en la interacción con otros personajes (a mayor trabajo en equipo, mayor interacción con otros personajes). Define el problema de asociación heurística como una red bayesiana expresando en ella al menos las relaciones indicadas no solo en este apartado sinó en el resto del enunciado, de forma que todas las características abstractas del problema que hayas definido en el apartado anterior tengan algun tipo de influencia en la solución. Separa bien en el diagrama que variables describen características de problema y cuales describen soluciones. Lista de forma clara los diferentes valores que puede tomar cada variable. Da un ejemplo de tabla de probabilidad de algún nodo, inventándote las probabilidades, pero expresando como influyen los valores de los nodos padre en las probabilidades de los valores de los nodos hijo.

Esta claro que en el gráfico deberíamos tener al menos dos grupos de nodos, los que corresponden a las características abstractas del jugador que se obtienen en la fase de abstracción, y los que corresponden a las características de la solución abstracta. Nuestro objetivo es construir una red que conecte características del problema abstracto a características de la solución abstracta.

La figura a continuación corresponde a la solución propuesta, que está alineada con las características abstractas del apartado anterior. Los nodos de la solución serán los nodos finales de la red bayesiana (en este caso cinco nodos con los 5 tipos de ajuste de los que consta la solución abstracta). Los nodos a su izquierda son las 7 características abstractas que modelan al jugador. En este caso no ha hecho falta hacer ninguna adaptación especial de las características del apartado anterior, ya que todas ellas tenian ya un rango de valores discretos.

Respecto a las dependencias, representaremos las dependencias entre nodos, ya sean de la solución o del problema abstracto. Es muy importante representar en la red cómo dependen los nodos solución de los nodos del problema abstracto, ya que ese es el objetivo principal de la fase de asociación heurística.



Una tabla de probabilidad simplemente ha de asignar más probabilidad a valores más correlacionados entre grupos de variables. Por ejemplo, si escogemos una variable como Ajustar_Nivel_IA_Enemigos (que depende de Nivel_Salud, Tiempo_Reacción y Tipología_Juegos), la tabla de probabilidad podría ser algo como lo siguiente:

			Ajustar_Nivel_IA_Enemigos		
Nivel_Salud	Tiempo_Reacción	Tipología_Juegos	incrementar	mantener	reducir
muy baja	lento	may_ingenio	0,0	0,01	0,99
muy baja	lento	variado	0,0	0,03	0,97
muy baja	lento	may_competitivo	0,0	0,04	0,96
muy baja	lento	may_acción	0,0	0,05	0,95
⋮	⋮	⋮	⋮	⋮	⋮
baja	lento	may_acción	0,0	0,2	0,8
⋮	⋮	⋮	⋮	⋮	⋮
normal	normal	may_ingenio	0,1	0,9	0,0
⋮	⋮	⋮	⋮	⋮	⋮
alto	rápido	may_competitivo	0,95	0,05	0,0
alto	rápido	may_acción	0,99	0,01	0,0

La tabla intenta reflejar, por ejemplo, que en igualdad de nivel de salud y tiempo de reacción, el nivel de IA del juego será algo mayor en jugadores cuya tipología de juego sea mayormente de acción, y va decreciendo progresivamente en el caso de mayormente competitivos, variados y mayormente ingénio. También refleja que a mayor salud y tiempo de reacción más alto, maror el incremento de la IA, y a menor salud y peor tiempo de reacción, la reducción del nivel de IA aumenta. Es importante asegurarse que la suma de los valores de cada fila de la tabla sumen 1.

2. (5 puntos) La empresa *TraeMeLo* quiere convertirse en la lider dentro del sector emergente de empresas que

reciben pedidos de sus clientes para ir a recoger algun producto/compra en uno de los comercios registrados en el sistema (normalmente una tienda, una farmacia, un restaurante) y traérselo a sus casas, siempre que sea dentro de la misma ciudad, no exceda los 10kg de peso y con un volumen de máximo 40x40x30cm. Los pedidos los llevan unos mensajeros, que pueden ir en bici (con un cajón trasero en el que pueden llevar un máximo de 2 pedidos a la vez) o en scooter (con un cajón trasero en el que pueden llevar un máximo de 4 pedidos a la vez).

Los clientes pueden solicitar a través de una app la recogida de un pedido en un comercio registrado (la app envia un identificador de comercio, y el sistema tiene un método auxiliar para convertir ese string en coordenadas de GPS) y la entrega en la dirección registrada del cliente (hay que tener en cuenta que un cliente puede realizar más de un pedido, y en una misma dirección puede haber más de un cliente). El sistema entonces queda a la espera a que el comercio envie un aviso de que el pedido está listo para ser recogido, y en ese momento asigna alguno de los mensajeros que tenga espacio dentro del cajón trasero para recoger el pedido en el comercio y llevarlo a su destino, pudiendo parar por el camino en otros puntos para recoger o entregar pedidos. La empresa nos pide que desarrollemos un planificador simple que genere un plan para recoger y entregar los pedidos recibidos en un periodo de tiempo con los mensajeros disponibles en cada momento (a veces una averia o una baja por enfermedad hace que ese número varie). En el modelo deben quedar explícitos los pedidos a recoger, el aviso del comercio de que el pedido está listo, la recogida del pedido por parte del mensajero y su entrega al cliente.

a. (3.5 puntos) Describe el dominio (incluyendo predicados, acciones, etc...) usando PDDL. Da una explicación razonada de los elementos que has escogido. Ten en cuenta que el modelo del dominio ha de poderse extender a más o menos mensajeros, más o menos comercios, más o menos clientes y a futuros vehículos capaces de llevar más pedidos a la vez.

Existen muchas posibles formas de modelar este dominio en PDDL (con más o menos predicados, con más o menos tipos, con más o menos operadores...) y por ello tomaremos decisiones que vayan encaminadas a crear un modelo que no contenga ineficiencias innecesarias.

Algunas cosas que tendremos en cuenta en la solución propuesta:

- intentaremos minimizar el número de operadores y el factor de ramificación de los mismos, de forma que se reduzca la exploración de que operadores son aplicables en cada momento;
- evitaremos operadores con parámetros similares, de forma que la existencia de unos objetos u otros dirija la instanciación de operadores;
- usaremos tipos en las variables para reducir en lo posible la cantidad de objetos que el planificador comprobará para cada parámetro del operador;
- intentaremos evitar el uso de `exists` y `forall` en las precondiciones y efectos de los operadores, ya que tienen un impacto negativo en el tiempo de cómputo y, en la práctica, aumentan el factor de ramificación por la existencia de variables ocultas (variables a instanciar que no son parámetros) dentro del operador;
- será más importante que la ejecución sea eficiente, aunque la representación sea más compleja (es decir, añadiremos predicados, operadores y tipos si eso puede facilitar la labor del planificador).

Una solución muy equilibrada sería la siguiente:

```
(define (domain Traemelo)
  (:requirements :adl :typing)

  (:types pedido lugar vehiculo cliente - object
          direccion slot - lugar)

(:predicates
   (estacionado ?ve - vehiculo ?d - direccion)
   (dentro ?s - slot ?ve - vehiculo)
   (libre ?s - slot)
   (pedido ?p - pedido ?c - cliente ?d - direccion)
   (en ?p - pedido ?l - lugar)
   (pendiente ?p - pedido ?d - direccion)
   (listo ?p - pedido)
   (servido ?p - pedido)
  )

  (:action preparar_pedido
    :parameters (?pe - pedido ?d - direccion)
```

```
    :precondition (pendiente ?pe ?d)
    :effect (and (listo ?pe) (en ?pe ?d) (not (pendiente ?pe ?d)))
  )

  (:action recoger_pedido
    :parameters (?pe - pedido ?ve - vehiculo ?s - slot ?d - direccion)
    :precondition (and (listo ?pe) (en ?pe ?d)
                       (estacionado ?ve ?d)
                       (dentro ?s ?ve) (libre ?s)
                   )
    :effect (and (en ?pe ?s) (not (en ?pe ?d))
                 (not (listo ?pe)) (not (libre ?s))
            )
  )

  (:action entregar_pedido
    :parameters (?pe - pedido ?c - cliente ?ve - vehiculo ?s - slot ?d - direccion)
    :precondition (and (en ?pe ?s) (dentro ?s ?ve)
                       (estacionado ?ve ?d)
                       (pedido ?pe ?c ?d)
                  )
    :effect (and (en ?pe ?d) (libre ?s) (servido ?pe)
                  (not (en ?pe ?s))
                  )
  )

  (:action ir_a
    :parameters (?ve - vehiculo ?ori - direccion ?des - direccion)
    :precondition (estacionado ?ve ?ori)
    :effect (and (estacionado ?ve ?des) (not (estacionado ?ve ?ori))
            )
  )
)
```

En esta solución se distingue entre clientes (los usuarios del servicio), pedidos (un identificador univoco de cada pedido), vehículos (los vehículos disponibles para el reparto), los slots dentro del cajón trasero de carga (cada slot es un "trozo" de cajón en el que podemos poner un pedido) y dirección (allí donde se han de recoger o entregar pedidos). Se ha decidido no modelar a mensajeros y vehículos por separado, solo a los vehículos, ya que cada vehículo tendrá un mensajero y solo uno, y no varía durante el reparto. Tampoco se ha modelado el peso y el tamaño de los pedidos o el peso máximo que puede llevar el vehículo, ya que se puede ver en el apartado b) que no se dispone de esa información para cada pedido, por lo que asumiremos que un pedido cumple las restricciones de peso y tamaño para poder ser colocado dentro de un slot en el cajón trasero de carga. El tipo lugar se ha creado para poder usar polimorfismo en el predicado `en`, y usar un solo predicado para decir que un pedido o bien está en una dirección o bien en el slot dentro de un vehículo. Modelamos las diferentes capacidades de cada vehículo (bici, moto) asignando slots disponibles a un vehículo dado, sin necesidad de tener tipos aparte para bicis y motos. Es importante recordar que solo el identificador de pedido es unívoco: el cliente puede realizar más de un pedido, una dirección de recogida puede preparar más de un pedido, y una dirección de entrega no solo puede recibir más de un pedido, sino que puede tener más de un cliente.
La lista de predicados es la siguiente:

- `en`, un predicado polimórfico que nos dice en que lugar (ya sea una dirección o un slot) está el pedido en todo momento, desde el momento en el que se prepara en la tienda.
- `estacionado`, que nos dice en que dirección está parado el vehículo en cada momento. Se ha creado un predicado nuevo en vez de aumentar el polimorfismo del predicado `en` para que se pueda aplicar a vehículos ya que los vehículos no pueden estar en cualquier lugar, solo en direcciones.
- `dentro`, que nos dice que un slot esta dentro de un cierto coche.
- `libre`, que nos dice que el slot no esta ocupado por ningun pedido. Podríamos usar el predicado `en` para saber si un slot tiene ya algun pedido, pero para saber si un slot esta libre o no tendríamos

que comprobar que para todos los pedidos en el modelo del problema no hay ningun pedido que esté `en` ese slot (y esto implica el uso de `forall` o de `(not exists)`). Por ello es más efectivo añadir el predicado `libre`, que sabiendo el slot nos dice si está libre o no.

- `pedido`, que nos da parte de la información del pedido: su identificador, el cliente que lo pide y la dirección de entrega. El resto de la información del pedido se codifica con el predicado `pendiente`.
- `pendiente`, que completa la información de un pedido dando su dirección de recogida, y sirve también para decir que un pedido está pendiente de ser creado en esa dirección. Se usa como filtro para restringir los pedidos que la precondición del operador `preparar_pedido` intenta explorar (una vez el pedido ha sido preparado ya no está pendiente y no se va a volver a considerar como candidato a ser preparado).
- `listo`, que nos dice que un pedido ya ha sido preparado y está listo para ser recogido. Para saber donde recogerlo hay que usar el predicado `en` sobre ese pedido. Se usa como filtro para restringir los pedidos que la precondición del operador `recoger_pedido` intenta explorar (una vez el pedido ha sido recogido, ya no está listo para ser recogido y no se va a volver a considerar como candidato a ser recogido).
- `servido`, que nos dice si un pedido ya ha sido entregado en la dirección de entrega. El plan acabará cuando todos los pedidos estan servidos (que es una condición más fácil de comprobar que mirar, para todo pedio, que esta en una dirección d y que esa dirección es la de entrega). Es importante remarcar que en este modelo `servido` no se puede substituir por `(not listo)`, ya que `pendiente` no solo nos da uno de los estados del pedido (pendiente de ser creado) sino también parte de la información del mismo, y necesitamos dos predicados para modelar los tres estados restantes del pedido: listo para ser recogido, ya recogido pero no entregado, y pedido entregado.

Hay dos cosas relevantes a comentar sobre los predicados. En vez de añadir un predicado `disponible` para indicar los vehículos que están disponibles y los que no (ya sea por averia o porque el mensajero está enfermo), se ha optado por declarar en el fichero de problema solo aquellos vehículos que estan disponibles. Y el reparto de toda la información del pedido en dos predicados en vez de tener un único predicado no es casual: `pedido` tiene solo los datos que necesita el operador `entregar_pedido` (cliente y dirección de entrega), mientras que `pendiente` aporta la dirección de recogida que necesita el operador `recoger_pedido`.

Se ha simplificado el problema a un modelo con solo cuatro operadores: `preparar_pedido` `recoger_pedido`, `entregar_pedido` e `ir_a`, que son autoexplicativos. El movimiento de los vehículos por la ciudad se ha simplificado a lo mínimo necesario, de forma que `ir_a` hace que el vehículo se mueva de una dirección a otra, sin modelar el camino que sigue. Las precondiciones se han colocado de forma que minimicen los objetos que se prueban (por ejemplo, tal como hemos dicho antes el operador `recoger_pedido` solo explora pedidos con el predicado `listo` a cierto). En el operador `ir_a` no se han colocado restricciones (moverse solo a direcciones con pedidos listos para recoger o que son dirección de entrega de un pedido dentro del vehículo) ya que hemos de permitir también que los vehículos puedan ir de vuelta al parking de la empresa. Es posible que para tamaños de problema más grandes se debería restringir el operador, poniendo algun tipo de restriccion en las rutas entre direcciónes para reducir el factor de ramificación, pero con cuidado de no restringir los movivimentos solo a pedidos para permitir que el vehículo vuelva al parking.

Es importante remarcar que este modelo permite modelar a más o menos vehículos, más o menos comercios, más o menos clientes y a futuros vehículos capaces de llevar más pedidos a la vez, tal y como pide el enunciado.

b. (1.5 puntos) El jefe del proyecto nos ha proporcionado una tabla como ejemplo de las peticiones que el sistema recibirá en un periodo de 20 minutos, con una serie de pedidos que se han de transportar desde un comercio registrado hasta la dirección del cliente. También nos dice que al inicio de ese intervalo de 20 minutos tenemos disponibles dos motos (capacidad máxima de 4 pedidos) y una bici (capacidad máxima de 2 pedidos) estacionadas en el parking central de *TraeMeLo*, y que tras entregar todos los pedidos del periodo las motos y la bici han de volver a ese parking.

pedido	dirección recogida	dirección entrega	cliente
p1	PizzaHotBalmes88	PauClaris55	Mario
p2	TodoPastaAragon167	Meridiana131	Aitana
p3	SupermercadoBruc31	Villaroel21	Pau
p4	frankfurtCasp13	ManuelGirona86	Amalia
p5	MercahoyValencia216	RamblaPrim12	Jaume
p6	Farmacia24hDiagonal340	Meridiana131	Aitana
p7	TodoPastaAragon167	Lepant95	Nuria
p8	FarmaciaGarciaSardenya501	Lepant95	David
p9	PizzaHotBalmes88	RocBoronat01	Esther
p10	BurgerFastCiencies62	AvCarrilet73	Miquel
p11	Farmacia24hDiagonal340	AvCarrilet73	Miquel
p12	MercahoyValencia216	RamblaPrim12	Laura

Describe este problema usando PDDL. Da una breve explicación de cómo modelas el problema.

En este caso el modelado del problema está totalmente marcado por el modelado del dominio del apartado anterior. El resultado es el siguiente:

```
(define (problem Traemelo-3vehiculos-12pedidos-10usuarios)
   (:domain Traemelo)
   (:objects p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 - pedido
             Mario Aitana Pau Amalia Jaume Nuria David Esther Miquel Laura - cliente
             Moto1 Moto2 Bici3 - vehiculo
             s11 s12 s13 s14 s21 s22 s23 s24 s31 s32 - slot
             parking PizzaHotBalmes88 Farmacia24hDiagonal340 TodoPastaAragon167
             SupermercadoBruc31 frankfurtCasp13 MercahoyValencia216
             FarmaciaGarciaSardenya501 BurgerFastCiencies62
             PauClaris55 Villaroel21 Lepant95 Meridiana131 RamblaPrim12
             AvCarrilet73 ManuelGirona86 RocBoronat01 - direccion
   )


(:init
  (pedido p1 Mario PauClaris55) (pendiente p1 PizzaHotBalmes88)
  (pedido p2 Aitana Meridiana131) (pendiente p2 TodoPastaAragon167)
  (pedido p3 Pau Villaroel21) (pendiente p3 SupermercadoBruc31)
  (pedido p4 Amalia ManuelGirona86) (pendiente p4 frankfurtCasp13)
  (pedido p5 Jaume RamblaPrim12) (pendiente p5 MercahoyValencia216)
  (pedido p6 Aitana Meridiana131) (pendiente p6 Farmacia24hDiagonal340)
  (pedido p7 Nuria Lepant95) (pendiente p7 TodoPastaAragon167)
  (pedido p8 David Lepant95) (pendiente p8 FarmaciaGarciaSardenya501)
  (pedido p9 Esther RocBoronat01) (pendiente p9 PizzaHotBalmes88)
  (pedido p10 Miquel AvCarrilet73) (pendiente p10 BurgerFastCiencies62)
  (pedido p11 Miquel AvCarrilet73) (pendiente p11 Farmacia24hDiagonal340)
  (pedido p12 Laura RamblaPrim12) (pendiente p12 MercahoyValencia216)
  (dentro s11 Moto1) (libre s11)
  (dentro s12 Moto1) (libre s12)
  (dentro s13 Moto1) (libre s13)
  (dentro s14 Moto1) (libre s14)
  (dentro s21 Moto2) (libre s21)
  (dentro s22 Moto2) (libre s22)
  (dentro s23 Moto2) (libre s23)
  (dentro s24 Moto2) (libre s24)
  (dentro s31 Bici3) (libre s31)
  (dentro s32 Bici3) (libre s32)
  (estacionado Moto1 parking)
  (estacionado Moto2 parking)
  (estacionado Bici3 parking)
)

(:goal (and (forall (?p - pedido) (servido ?p))
```

```
                (forall (?v - vehiculo) (estacionado ?v parking))
        )
)

)
```

Se ha creado un objeto por cada pedido, cada cliente, cada vehículo, cada slot disponible en los vehíoculos y cada dirección (ya sea de recogida, de entrega o el parking de la empresa). Los nombres intentan hacerlo más facil de leer, pero al planificador le da lo mismo el nombre escogido para cada objeto.
El estado inicial es una fotografia de la configuración que aparece en el enunciado, indicando para cada pedido su cliente y su dirección de entrega, que está pendiente de ser recogido en una dirección, la pertenencia de cada slot a un vehículo y la posición inicial de los vehículos (en el parking de la empresa).
El estado objetivo usa el predicado `servido` para los pedidos (el plan acaba cuando todos los pedidos han sido servidos por los vehículos disponibles). Se podría haber descrito listando los doce predicados `servido` (uno por pedido), pero en este caso se ha optado por la expresión `forall` en adl, ya que nos permite cambiar el número de pedidos del problema sin tener que cambiar la descripción del estado objetivo, y no nos añade complejidad (en FastForward el `forall` en los objetivos se instancia una sola vez generando los predicados individuales, con un coste lineal respecto al número de objetos a explorar). Lo mismo se ha hecho para indicar que todos los vehículos han de estar estacionados en el parking en el estado final. Aunque con solo tres vehículos podríamos escribir fácilmente los tres predicados `estacionado` instanciados, de nuevo el uso del `forall` permite modificar los vehículos del problema sin tener que modificar la expresión del objetivo.

Las notas se publicarán el día **4 de julio**.