Universidad Carlos III de Madrid
Escuela Politécnica Superior

Proyecto Fin de Carrera
Ingeniería Técnica en Informática de Gestión

Ampliación y mejora del universo virtual AI-LIVE

Autor: Javier Escudero Moreno
Tutores: Javier Ortiz Laguna
Daniel Pérez Pinillos
Año: 2011

ÍNDICE

Índice de ilustraciones

Índice de tablas

1 INTRODUCCIÓN

En este apartado se ofrece una pequeña introducción al proyecto realizado y la estructura general del resto del documento.

1.1 Ámbito del proyecto

La Inteligencia Artificial (IA o AI del inglés, "Artificial Inteligence") es la rama de la informática que trata de obtener sistemas artificiales que se comporten o emulen ciertos aspectos de la inteligencia humana. El propósito de estos sistemas es resolver problemas o comportarse de una forma semejante a como lo hace un ser humano.

Este proyecto trata de introducir nuevos aspectos en la simulación sobre una arquitectura ya planteada anteriormente y la generación de nuevos comportamientos de los personajes simulados por los clientes que se conectan a la arquitectura.

El sistema sobre el que se ha trabajado es el videojuego AI-LIVE que es un videojuego de simulación social creado con el objetivo de simular situaciones del mundo real con personajes controlados de forma autónoma. En el videojuego, los distintos personajes llamados actores toman decisiones basadas en la situación del escenario donde se encuentran y tienen unos objetivos marcados que rigen estas decisiones.

El juego está basado en la arquitectura cliente-servidor, donde cada actor es controlado por un cliente distinto, bien por un motor de IA, o por un humano y el servidor que se encarga del control del escenario y ejecución de las acciones y decisiones que toman los clientes. El servidor otorga de forma ordenada el turno de ejecución a cada cliente. Además existe un motor gráfico para visualizar los escenarios y distintos actores conectados al servidor.

1.2 Objetivos

El objetivo del proyecto es mejorar algunos aspectos de la aplicación AI-LIVE desarrollada anteriormente por otros alumnos.

A grandes rasgos se van a eliminar algunas restricciones del universo virtual como el número de clientes conectados simultáneamente, haciéndolo independiente de los objetos iniciales en un escenario y cambiar algunos aspectos como el indicador de felicidad de los actores.

También se planea ampliar el proyecto añadiendo nuevas funcionalidades como el concepto de tiempo, nuevos objetos y acciones disponibles para los actores y mejorar algunos aspectos como la compilación y ejecución de los distintos módulos y centralizar los archivos de configuración que definen los atributos de los personajes disponibles que se llaman perfiles.

1.3 Organización de la memoria

Este documento describe la realización del proyecto de final de carrera y está estructurado en los siguientes apartados:

- **Apartado 1: Introducción**
  Descripción general del proyecto y visión de la estructura de la memoria.

- **Apartado 2: Estado del arte**
  Descripción de tecnologías que se han utilizado en el proyecto y otras alternativas que se podrían haber usado. Se describen distintos tipos de videojuegos y algunas de sus técnicas de inteligencia artificial empleadas. Se analizan distintas herramientas de desarrollo de software, técnicas de inteligencia artificial, lenguajes de script para automatizar procesos y distintos sistemas software de control de versiones.

- **Apartado 3: Objetivos del proyecto**
  Resumen de los objetivos que se querían alcanzar con la realización del proyecto.

- **Apartado 4: Gestión del proyecto**
  Explicación de la metodología empleada en la realización del proyecto. Se exponen tareas en las que se ha dividido junto a la duración y coste de las mismas.

- **Apartado 5: Memoria del trabajo realizado**
  Descripción del trabajo realizado. Descripción de los distintos módulos, protocolo de comunicaciones y el modelo de conocimiento de la aplicación.

- **Apartado 6: Evaluación de la aplicación.**
  Pruebas realizadas y evaluación de resultados de las pruebas.

- **Apartado 7: Manual de usuario**

Breve manual de uso de la aplicación describiendo como ejecutar cada módulo y las distintas opciones de configuración.

- **Apartado 8: Conclusiones**
  Conclusiones obtenidas con la realización del proyecto.

- **Apartado 9: Trabajos futuros**
  Exposición de posibles mejoras para ampliar el proyecto.

- **Apartado 10: Bibliografía**
  Bibliografía consultada durante la realización del proyecto.

- **Apartado 11: Referencias**
  Enlaces web consultados durante la realización del PFC.

- **Apartado 12: Anexos**
  Documentos anexos a la memoria que se adjuntan como ayuda al código fuente. El manual de usuario que contiene indicaciones de los requisitos para ejecutar los distintos módulos de la aplicación y el manual de referencia que indica detalladamente todos los módulos para facilitar la ampliación y mantenimiento de la aplicación.

2 ESTADO DE LA CUESTIÓN

A continuación se realiza un repaso al estado actual de la tecnología utilizada por AI-LIVE y videojuegos parecidos a este proyecto que usan técnicas de inteligencia artificial. Además se describirá la herramienta para crear el motor de inteligencia artifical utilizada en el proyecto que es *CLIPS*.

2.1 Los videojuegos

Los videojuegos son programas informáticos de carácter lúdico en el que uno o varios usuarios interaccionan con el sistema.

Desde el primer videojuego con éxito comercial (1972, *PONG*, desarrollado por Atari) se hizo necesaria la simulación de cierta inteligencia en los juegos o por lo menos una respuesta coherente a las acciones de los jugadores. En ese mismo videojuego, que simula un partido de tenis en el que un jugador compite contra otro, controlando una especie de paleta para golpear una pelota e introducirla en el campo del jugador contrario, se programó cierta inteligencia para que el oponente controlado artificialmente pudiera devolver la pelota contra el campo del jugador humano. En principio la "inteligencia" consistía en calcular la posición de la pelota y desplazar la paleta para alcanzarla.

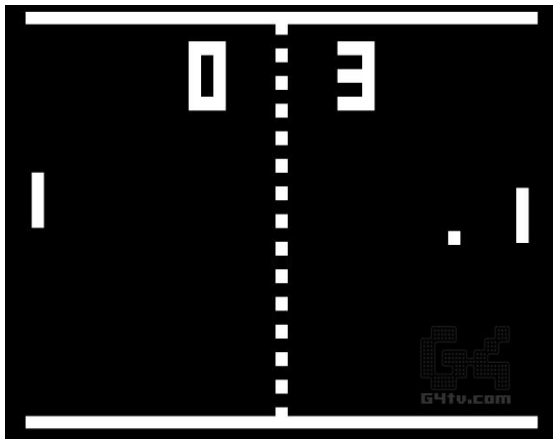


Ilustración 1: Videojuego "Pong"

Posteriormente salieron muchas versiones de este videojuego y otras variaciones que no sólo modificaban su jugabilidad, sino actualizando la inteligencia artificial de los oponentes y simulando distintas técnicas para intentar superar al oponente humano, como en *Shufflepuck Cafe* en el que el nivel de dificultad es distinto en cada

oponente y cada uno utiliza una serie de movimientos característica con una estrategia definida para desafiar al jugador.

Ilustración 2: Videojuego Shufflepuck Cafe

2.1.1 Tipos de videojuegos

Hay una infinidad de videojuegos distintos clasificables por un género en particular, aunque muchos videojuegos se pueden englobar en más de un género por tener distintas fases que emplean las técnicas propias de cada género. Una posible clasificación puede ser:

- **Acción**: requieren al jugador reflejos y precisión. Dentro de este género se pueden encontrar los subgéneros Bet 'em up, juegos en primera persona, puzzles o plataformas.

- **Aventura**: estos videojuegos narran una historia y ofrecen un reto al jugador mediante la resolución de ciertos puzzles y la interacción con otros personajes, generalmente sin confrontaciones sino con diálogos para obtener información útil o determinados objetos esenciales para resolver los puzzles. Algunos juegos de este tipo son *Myst* o *The Monkey Island.*

- **Juegos de rol**: son videojuegos en los que el jugador controla a un personaje o grupo de personajes con ciertas

características y habilidades que pueden desarrollar a lo largo del videojuego con diversos enfrentamientos contra otros personajes.

- **Simulación**: es un género bastante amplio que trata de simular cierto aspecto de la realidad y recoge a su vez varios tipos de subgéneros como videojuegos de simulación de coches como *Gran Turismo* o *Need for Speed*, de construcción y gestión como *Sim City* o *Theme Park*, simuladores de vida como *Creatures*, *Eco* o *The Sims*.

- **Estrategia**: son videojuegos donde se controla una serie de unidades especializadas, luchando contra otras unidades y obteniendo recursos para mejorar las unidades para alcanzar la victoria.

2.1.2 Juegos Beat 'em up (FPS)

Estos son videojuegos en los que el jugador controla a un personaje y realiza ataques a corta distancia con las manos desnudas o con pequeñas armas como cuchillos o espadas. El jugador lucha contra un gran número de enemigos más débiles durante varias fases y otros videojuegos del género se centran en peleas orientadas al uno contra uno con enemigos de dificultad balanceada. Algunos ejemplos de juegos de este tipo son *Golden Axe* o *Street Fighter*.

Ilustración 3: Golden Axe

Ilustración 4: Street Fighter

Los videojuegos de lucha orientados a peleas uno contra uno suelen tener un repertorio de movimientos disponibles bastante grande con ataques de distinto alcance y poder para dañar al adversario. Los enemigos controlados por la IA del juego suelen tener un patrón de movimiento y realizan distintas acciones dependiendo de los ataques y movimientos que realiza el jugador.

Ilustración 5: Tekken

2.1.3 Juegos de disparos en primera persona (FPS)

Otro tipo de videojuegos como los llamados *first person shooter* o videojuegos de disparos en primera persona, donde el jugador tiene una visión en primera persona de su personaje y tiene a su

disposición una serie de armas que utiliza contra el resto de oponentes u objetivos. Estos videojuegos tienen que dotar de cierta inteligencia a los enemigos para ofrecer un reto al jugador. Algunos ejemplos de este tipo de videojuegos son *Doom*, *Half-Life*, la serie *Unreal* y *Unreal Tournament* de la compañía Epic o *F.E.A.R.*

2.1.3.1 Doom

Doom es una serie muy popular de videojuegos de este género desarrollada por Id Software. La serie se centra en un personaje que lucha contra hordas de enemigos para sobrevivir. Se considera como el pionero en este género ofreciendo gráficos en tres dimensiones y tecnología multijugador en red.

Esta serie de videojuegos se caracteriza por un gran número de armas disponibles de distintos tipos agrupadas en clases que han servido de base a multitud de videojuegos del género, como pistolas, escopetas, armas de corta distancia, ametralladoras o armas de mayor calibre. También se caracteriza por las altas dosis de violencia en algunas de sus escenas, elementos que se utilizan en otros videojuegos del género.

Ilustración 6: Doom

2.1.3.2 HalfLife

Esta serie de videojuegos de Valve está enfocada en la historia del personaje principal del juego llamado *Gordon Freeman* y su historia creada específicamente para él. La IA del juego se concentra en los personajes secundarios y enemigos, que en su día fue bastante revolucionaria ya que en determinadas secuencias cinemáticas, el jugador podía interactuar con los otros actores y estos a su vez ayudaban y acompañaban al jugador realizando diversas acciones, atacando a los enemigos del jugador.

Los enemigos del juego a los que se enfrenta el jugador también son capaces de seguir cierta estrategia de ataque actuando como un grupo sincronizado o atacando por sorpresa.

La implementación de la IA se basa en un sistema de objetivos y un programador de tareas. Existen una serie de tareas predefinidas muy sencillas como desplazarse a un sitio, agacharse, esquivar, esconderse, etc. también existen una serie de condiciones de diversa índole como que el personaje pueda utilizar un ataque a corta distancia o que el enemigo está demasiado lejos y sobre estas premisas los enemigos deciden qué acción realizar.

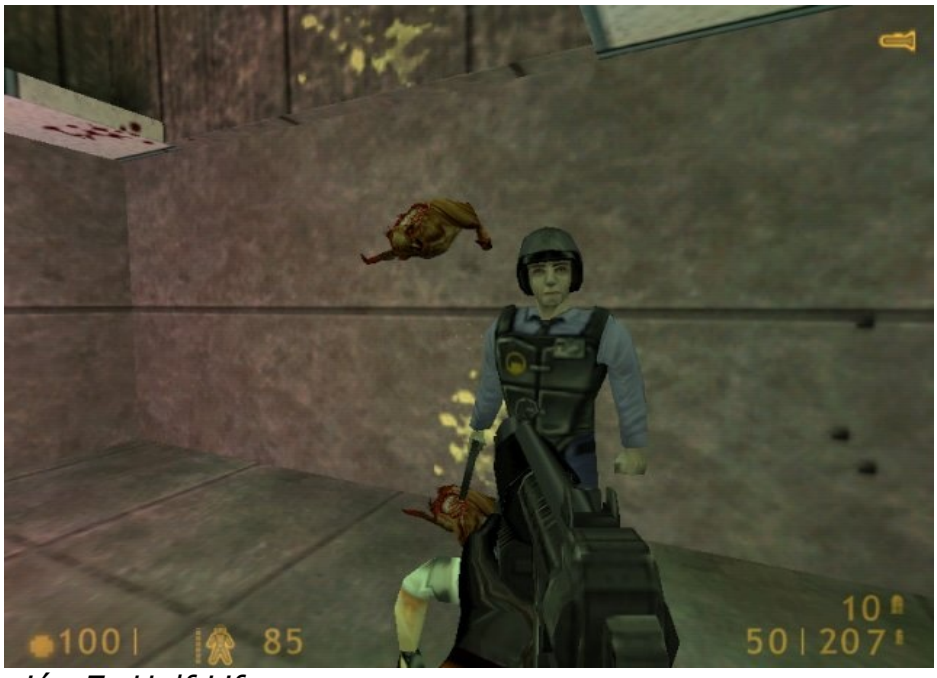
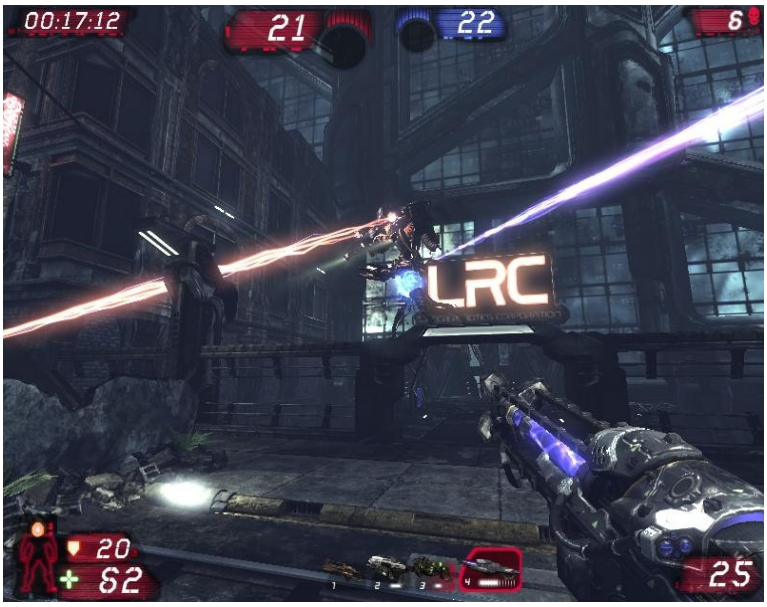


Ilustración 7: Half Life

2.1.3.3 Unreal

Unreal es una serie de videojuegos desarrollados por Epic Games y Digital Extremes. Tiene un motor gráfico llamado *Unreal Engine* que se ha desarrollado a lo largo de los años y empleado en videojuegos de la misma serie como *Unreal Tournament* o *Unreal Championship* y videojuegos de otras compañías como *Bioshock* y *Splinter Cell*. Tuvo mucho éxito gracias en parte a las posibilidades de expansión del juego mediante su propio lenguaje de script, el *Unreal script* y a la inclusión de un editor de escenarios. Estas facilidades para ampliar y mejorar el videojuego dieron lugar a extensas comunidades que aportaban diverso contenido al juego.

Ilustración 8: Videojuego Unreal tournament 3



El editor de escenarios permite crear o modificar un escenario determinado y también permite programar la inteligencia artificial de los oponentes. Se crean patrones de movimientos disponibles por el mapa para que los personajes controlados por la IA del juego se desplacen por todo el escenario y recojan diversos objetos. Para ello, se añade un tipo especial de objeto, no visible por los jugadores que enlaza distintos puntos del escenario y marca los caminos posibles que puede tomar un personaje autónomo.

Al crear un mapa no es necesario indicar y programar todos los movimientos de los personajes ya que aunque no hay que indicar en sí el movimiento o acciones concretas de los personajes controlados

por la máquina, que se mueven, disparan al resto de oponentes y recogen objetos a su alcance, deben cumplir ciertos objetivos en los distintos escenarios.

Para que los personajes controlados por la IA realicen acciones complejas para conseguir los objetivos del juego se emplean ciertos objetos en el editor de mapas, no visibles para el jugador y similares a los que indican los caminos posibles que pueden tomar los personajes, que indican a los actores controlados por la máquina o también llamados *bots* distintas acciones y objetivos como saltar en cierto punto, recoger un objeto y llevarlo a otro punto, atacar un objeto o defender cierta posición.

Bajo ciertas circunstancias, estos *bots* modifican su comportamiento de forma que si el jugador ataca la base principal por ejemplo, rápidamente dejan la actividad que estaban realizando y se dirigen a defender ese objetivo.

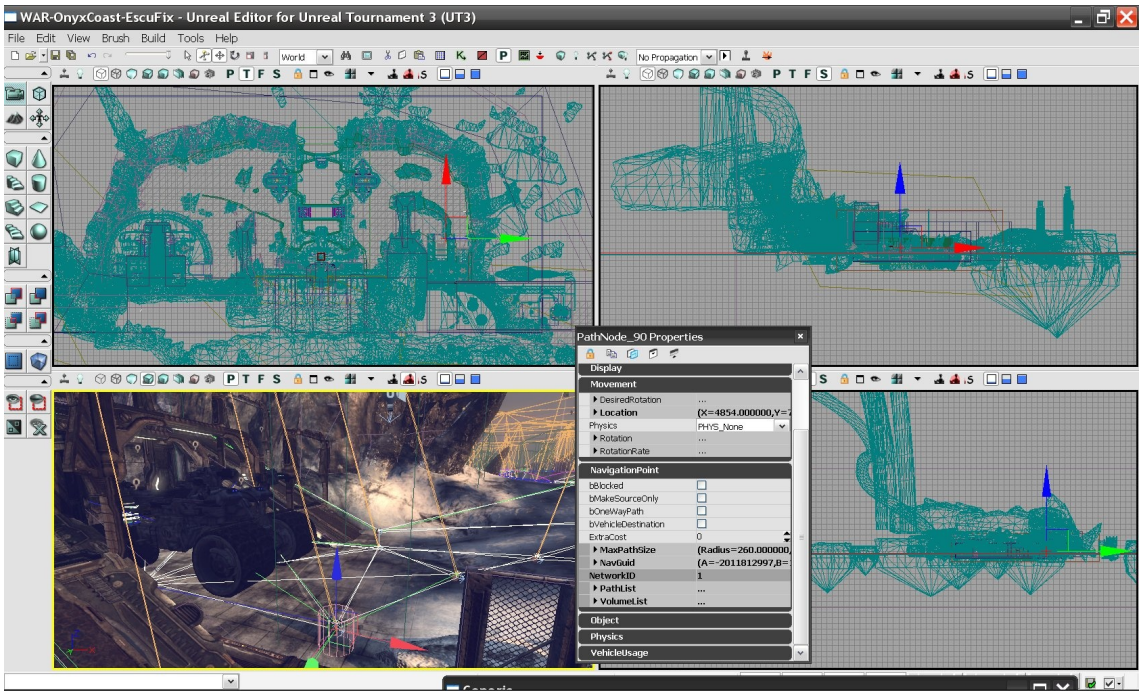


Ilustración 9: Editor UnrealEd

2.1.3.4 F.E.A.R

Otro ejemplo de este tipo de videojuegos pero con una inteligencia artificial en principio más compleja es *F.E.A.R.* desarrollado por Monolith Productions y en el que los enemigos son capaces de aprovecharse del escenario para protegerse y rodear al jugador e intentar acabar con él desde una posición segura.

La IA del juego está bastante simplificada, de forma que solamente se centra en dos aspectos de los personajes:

- **Animación**: realizar una animación completa que mueve al personaje como efecto secundario. La animación de cubrirse del ataque de otro personaje, conlleva implícitamente el desplazamiento hacia un objeto para situarse detrás de él.

- **Movimiento**: seleccionar y controlar la animación para dirigirse en una dirección determinada: se conoce la posición del jugador y se selecciona una dirección para encontrarse con él.

Las animaciones también tienen mensajes para controlar o activar otros sistemas como por ejemplo al andar, se activa el sonido de los pasos, de esta forma se simplifica aún más el motor de inteligencia artificial.

En este juego, al igual que en *Los Sims* se utilizan los llamados "objetos inteligentes" para implementar distintas animaciones cuando los actores interactúan con estos objetos. Cada objeto tiene asociadas una serie de acciones que se ejecutan de forma periódica o cuando un personaje interactúa con el objeto. Esta técnica se llama *microthreads* donde cada objeto tiene un tiempo de ejecución. De esta forma se obtiene un sistema modular mucho más fácil de extender, ya que se pueden incluir nuevos objetos y acciones sin cambiar el motor de IA realmente, sino programando las acciones en los propios objetos.

Ilustración 10: F.E.A.R

2.1.4 Puzzles

Los puzzles son videojuegos que ponen a prueba el ingenio de los jugadores resolviendo algún enigma o superando diversas fases evitando a varios enemigos u obstáculos. *Pac-Man* es un ejemplo muy famoso de este tipo de videojuegos en el que el objetivo consiste en recoger todos los objetos de una determinada clase repartidos por el escenario y evitar a su vez a los enemigos que eliminarán al jugador si consiguen hacer contacto con el personaje del jugador.

Ilustración 11: Pac-Man

2.1.5 Plataformas

Las plataformas son un subgénero dentro del género de acción en el que el personaje controlado por el jugador viaja a través de unos mundos virtuales creados con plataformas sobre las que saltar, evitando enemigos y otros obstáculos. Algunos ejemplos de este género son *Sonic*, *Mario* o *Crash bandicoot*.

Sonic se caracterizaba por una velocidad de movimiento en la pantalla bastante grande y diversos elementos y saltos que hacían al juego atractivo visualmente. La mayoría de este tipo de videjuegos cuentan con distintos objetos que el jugador puede recoger para incrementar la puntuación final del videojuego.

Ilustración 12: Sonic

Los enemigos de este tipo de vidojuegos suelen tener un comportamiento simple, con un movimiento prefijado o persiguiendo al jugador y en determinadas fases aparece un tipo especial de enemigo más poderoso y con distintos ataques y estrategias que el jugador debe vencer para pasar de fase y avanzar en el videojuego.

Ilustración 13: Crash bandicoot

En un principio estos videojuegos estaban desarrollados con gráficos en dos dimensiones y ofrecían un desplazamiento horizontal

del mundo virtual por el que se desplazaban los personajes. Después aparecieron los gráficos en tres dimensiones y nuevas alternativas en el desplazamiento de los escenarios como en profundidad como se aprecia en la Ilustración 13 de *Crash Bandicoot* u ofreciendo una completa libertad de movimientos por el escenario.

2.1.6 Aventuras

Este género de videojuegos engloba un gran número de videojuegos que suelen mezclar elementos de otros géneros como plataformas o puzzles. Algunos juegos de este tipo presentan una interfaz sencilla con varias acciones disponibles y el jugador debe realizar ciertos objetivos para avanzar en el juego. La mayoría tiene una historia predefinida que se va revelando al jugador mientras realiza una serie de acciones con los elementos del escenario y en el orden correcto, como por ejemplo buscar la llave de una puerta en una habitación para poder salir de ella.

Ilustración 14: Indiana Jones

Muchos videojuegos de este género están inspirados en películas como *Indiana* Jones de la ilustración 14 y otros son historias completamente originales y suelen tener referencias o guiños a otros juegos del género y grandes dosis de humor en ciertas situaciones que viven los personajes del juego.

Ilustración 15: Monkey Island

Monkey Island tiene varias situaciones irrisorias como en la ilustración 15.

2.1.7 Juegos de rol

Los juegos de rol, de sus siglas en inglés *Role Playing Game* son un género de videojuegos basados en los juegos de rol de mesa como *Dragones y mazmorras* o *Diablo*.

En este género los jugadores asumen el papel o rol de un determinado personaje con una serie de atributos y cualidades que pueden desarrollarse a lo largo de las distintas historias disponibles como aventura.

Estos videojuegos cuentan con una serie de fases claramente diferenciadas como son las fases de exploración del mundo virtual y fases de combate. En las as fases de exploración se desarrolla la historia, se desarrollan numerosos diálogos y el jugador obtiene información y pistas para lograr sus objetivos. En estas fases intervienen otros actores para contar fragmentos de la historia o temática en torno a la que gira el juego y se pueden alternar fases de combate con distintos enemigos. Algunos juegos interrumpen el trascurso de una fase activando otra como por ejemplo la transición de una fase de exploración al combate, ofreciendo algún efecto visual para resaltar este hecho e introduciendo una pausa en el desarrollo de la acción.

Otros juegos ofrecen una transición sin ninguna pausa en medio, simplemente activando alguna animación en el personaje y la interfaz de usuario ofreciendo indicando al jugador que se encuentra en una fase de combate.

Existen también diversas variaciones en el propio transcurso de cada fase, por ejemplo en las fases de combate, dándose casos en los que se realiza la acción por turnos, en los que los personajes controlados por el jugador disponen de un tiempo limitado para realizar distintos ataques u otras acciones. Existen variaciones en las que no se dispone de un sistema de turnos prefijados sino que cada acción tiene asociado un tiempo de preparación o recuperación del personaje para poder realizar la siguiente acción y el jugador va encadenando distintas acciones y ataques de forma mucho más dinámica y rápida.

Algunos ejemplos de estos videojuegos son la serie *Final Fantasy*, *Kingdom Hearts* o *Vagrant Story.*

2.1.7.1 Final Fantasy

Esta es una serie de videojuegos desarrollada por Square Enix y su principal género es el RPG, aunque han salido numerosas secuelas y versiones bajo el mismo nombre pero en otros géneros como carreras o lucha.

Los videojuegos de la serie centrados en el género RPG mantienen elementos en común como son diversos objetos o actores que aparecen en los videojuegos. También se mantienen elementos fácilmente distinguibles como el sistema de batalla que entra en juego cada vez que el grupo debe enfrentarse a algún enemigo mientras exploran el mundo del videojuego. En un principio se usaba el sistema de batalla por turnos pero posteriormente se implementó el llamado sistema de batalla con tiempo activo en el que cada actor tiene un medidor de tiempo que le indica en qué momento puede realizar una acción. El jugador puede indicar una acción para cada uno de sus personajes y los actores las realizarán en cuanto el medidor de tiempo se lo permita.

Los actores generalmente tienen una serie de comandos disponibles para realizar dependiendo de los objetos que tenga de equipo como armas, ataques mágicos o utilizando algún objeto. Los objetos los suelen obtener durante las fases de exploración y existen diversas clases como objetos de recuperación de energía, mejoras para los atributos de los personajes como un medidor de tiempo acelerado para realizar acciones más rápidamente u otros objetos de ataque para dañar a los enemigos.

Ilustración 16: Final Fantasy VII sistema ABS

Ilustración 17: Final Fantasy X sistema por turnos variables

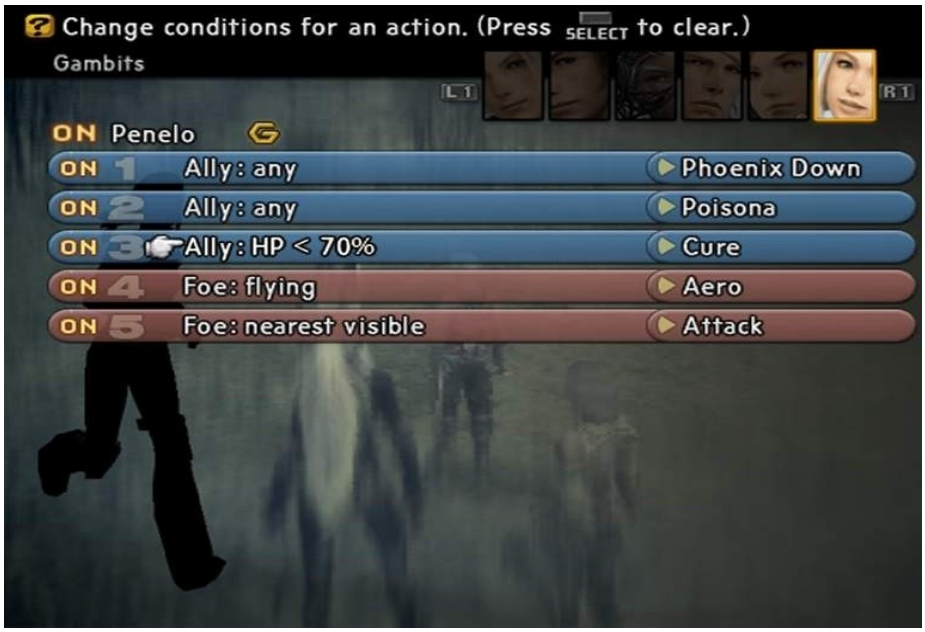
Los enemigos del juego son controlados por el motor de IA y generalmente tienen un patrón de ataque definido, empleando alguna estrategia como intentar acabar con el personaje más débil o emplear algún objeto o ataque que impida a los actores realizar acciones normalmente. Algunos efectos adversos que afectan a los personajes que controla el jugador impiden el movimiento durante algunos turnos a un personaje, pueden provocar que ataque a los miembros

de su propio equipo o disminuyen la precisión de los ataques de un personaje. Estos efectos adversos también puede inducirlos el jugador en los enemigos, proporcionando así un gran abanico de posibilidades para salir victorioso en los enfrentamientos.

El sistema de batalla ha evolucionado a lo largo de los años y han implementado sistemas de batalla en tiempo real o presentan modificaciones en el sistema de batalla con tiempo activo. Por ejemplo en *Final Fantasy X* se implementó un sistema de batalla por turnos condicionales en el que se volvía al sistema de batalla por turnos pero introduciendo variaciones en los mismos dependiendo de las acciones que selecciona el jugador para cada personaje, ofreciendo así algunas dosis de estrategia a la hora de seleccionar el tipo de ataque a realizar.

En *Final Fantasy XII* se introdujo un sistema de inteligencia artificial en el que varios actores del grupo protagonista son controlados por el motor de IA del juego de forma autónoma, pero respondiendo a una serie de condiciones que configura el jugador y que puede variar en cualquier momento. Por ejemplo si uno de los actores es capaz de lanzar hechizos de ataque a enemigos y curar a otros personajes del grupo protagonista, se indican las condiciones de atacar a un enemigo vulnerable a cierto hechizo en cuanto sea posible pero curar antes a un compañero del grupo para evitar que muera en la batalla.

Ilustración 18: Final Fantasy XII condiciones IA



2.1.7.2 Kingdom hearts

Kingdom Hearts es una serie de videojuegos desarrollada por Square Enix y Disney Interactive Studios y se puede clasificar como "Action RPG" ya que mezcla elementos típicos de los RPG con un sistema de experiencia y desarrollo de los personajes, diversos objetos y equipo con fases de exploración y batallas mientras se desarrolla la historia que no interrumpen la acción del juego. También se incluyen algunos elementos de otros géneros como las plataformas para alcanzar algunos puntos de los escenarios. Los elementos de acción los toma en las fases de lucha o batallas contra los enemigos en los que se ofrece una libertad de movimiento total al jugador mientras que puede desplazarse por todo el escenario y seguir realizando otras acciones distintas como ponerse a salvo lejos del alcance de los ataques del enemigo, recoger objetos del escenario o apoyar a otros personajes.

Ilustración 19: Kingdom Hearts

En este videojuego, el jugador controla al personaje principal y obtiene el apoyo de otros dos personajes controlados por la IA del juego que realizan ataques, acciones evasivas o defensivas y apoyo al resto de personajes proporcionando objetos curativos o hechizos de ayuda diversos.

2.1.7.3 Vagrant Story

Este videojuego también desarrollado por la compañía Square combina elementos del RPG con el género de las plataformas y puzzles ya que el personaje debe desplazarse por un escenario lleno de obstáculos y resolver algunos puzzles para poder continuar con la aventura. Al contrario que otros juegos, el jugador solamente controla al personaje principal y no se alternan fases de exploración en las que interactuar con otros personajes sino que la historia se va desarrollando mediante escenas programadas mostrando ciertos hechos relevantes y nuevos objetivos para motivar al jugador.

Ilustración 20: Vagrant Story

El sistema de batalla requiere de cierta habilidad en el jugador para presionar los botones correctos en el momento de atacar para que de esta forma el personaje encadene sucesivos ataques, se defienda o también puede utilizar diversos hechizos curativos o para atacar a sus enemigos. Al comenzar la orden de atacar, aparece una retícula alrededor del personaje para indicar el alcance que tiene con el arma equipada y seleccionar distintas partes del cuerpo del enemigo a atacar.

En cuanto a los objetos y diverso equipo que puede portar el personaje en el videojuego como armas y armaduras, un rasgo clásico del género, en este juego en concreto se centra en un sistema de desarrollo de armas, recolectando diversos materiales y objetos

por los escenarios y los que dejan los enemigos al ser derrotados. Estos objetos y materiales son combinados después para obtener nuevas armas, reparar las ya obtenidas o dotarles de ciertas características especiales para causar más daño a unos enemigos u otros.

2.1.8 Juegos en línea

En la historia de los videojuegos, desde la expansión de Internet, se ha aprovechado la conectividad que ofrece la red entre jugadores de diversos puntos del planeta para crear modos de juego cooperativos o enfrentar a los jugadores entre sí. Algunos de los juegos mencionados anteriormente tienen esta posibilidad y en lugar de enfrentarse a una inteligencia artificial programada, los jugadores se enfrentan entre sí o contra varios jugadores desde distintos puntos del planeta. Este modo suele ser muy atractivo para los jugadores porque les supone un reto enfrentarse a oponentes capaces de improvisar o realizar comportamientos complejos.

Algunos juegos en línea ofrecen la posibilidad de formar equipos o avanzar en la historia que cuenta el videojuego mediante un modo cooperativo. Un ejemplo de este tipo de videojuegos es el mencionado *Unreal Tournament 3*.

Otro tipo de videojuegos con mucho éxito comercial son los llamados MMO, de sus siglas en inglés *Massively Multiplayer Online* o videojuego multijugador online masivo. Este tipo de videojuegos permiten la conexión de cientos de jugadores a un mismo universo virtual en el que cooperar entre ellos para realizar determinadas acciones o enfrentarse entre varios grupos numerosos de jugadores.

La principal característica de los MMO es que la compañía que lo desarrolla suele mantener un gran número de servidores en los que se aloja el mundo virtual en sí. Cada jugador se conectará a un servidor u otro dependiendo de su ubicación, carga y la disponibilidad de estos servidores pero todos mantienen la sincronización para que un jugador de Tokio esté desplazando a su personaje junto a otro de Londres en tiempo real.

Estos videojuegos suelen ofrecer a sus jugadores oponentes controlados por IA y bien por ser más numerosos o por ciertas características especiales, los jugadores humanos deben organizarse en grupos para enfrentarse a estos oponentes. En muchos de estos videojuegos existe una gran comunidad de jugadores online detrás que suelen agruparse y se conectan a la vez al videojuego, creando así grandes comunidades de usuarios.

Algunos ejemplos de este tipo de videojuegos son la serie *World of Warcraft*, *Final Fantasy online* o en el género de los simuladores sociales, *Second Life*.

Second Life ofrece un llamado "metaverso" que es un universo virtual persistente en el que los jugadores interactúan entre sí y con el mismo mundo virtual. Este mundo virtual ofrece un entretenimiento que se basa en el mundo real, ofreciendo conciertos, películas y diversas actividades. Muchas empresas lo utilizan como un elemento más de publicidad, tienen un terreno y edificios como en el mundo real, resultado ser una fuente más de publicidad.

Dentro de este tipo de juegos, se pueden dividir en varios tipos:

- **MMO RPG** o juegos de rol de sus siglas en inglés *Role Playing Game*. Son los juegos MMO más extendidos en los que se controla un personaje o grupo de personajes para completar una serie de aventuras, con fases de exploración por el mundo virtual, batallas contra enemigos cada vez más complicadas, elementos narrativos con conversaciones con otros personajes, adquisición de información, etc. *Final Fantasy XI*, *Guild Wars* y *World of Warcraft* son algunos ejemplos de juegos de este tipo.

- **MMO FPS**: son videojuegos de disparos en primera persona pero enfocados a grupos de jugadores sobre un mundo virtual persistente. Los juegos de este tipo tienen características similares a los anteriores como los puntos de experiencia para evaluar a los personajes. Algunos ejemplos de estos juegos son *PlanetSide* o *World War II Online*.

- **MMO RTS** de sus siglas *Real Time Strategy* o estrategia en tiempo real: son videojuegos en los que el jugador asume el papel del líder de un ejército en una batalla, dirigiendo a sus unidades contra el enemigo y a la vez manteniendo y administrando los recursos necesarios para llevar a cabo esos enfrentamientos, como crear nuevas unidades, armas o edificios. Algunos ejemplos de estos juegos son *Battleforge*, *WorldShift* y *Galaxy Online*.

- **MMO deportes**: los jugadores pueden competir en los deportes tradicionales de equipo como fútbol, baloncesto, hockey, béisbol o fútbol americano.

- **MMO carreras**: son videojuegos donde se compite en carreras con diversos vehículos. Algunos juegos de este tipo son *Trackmania*, *Test Drive Unlimited* o *Need For Speed World*.

- **MMO juegos musicales**: juegos de baile inspirados en la serie *Dance Dance Revolution*.

- **MMO juegos sociales**: son simuladores sociales en mundos virtuales gigantescos. Un ejemplo de este tipo es *Second Life*.

- **MMO simuladores del mundo real**: son mundos virtuales creados específicamente para simular ciertos entornos como simuladores de vuelo o batallas de tanques.

- **MMO juegos de estrategia por turnos**: en estos juegos existe un control del tiempo que marca a los jugadores el momento que tienen disponible para realizar diversas acciones con todas las unidades de sus ejércitos, permitiéndoles controlar un gran número de unidades de forma simultánea. Los turnos se conceden a los jugadores de igual forma que en algunos juegos de mesa como el ajedrez . Algunos ejemplos de este género son *UltraCorps* o *Darkwind: War on Wheels.*

2.1.9 Estrategia

Este género de videojuegos se caracteriza por ofrecer a los jugadores crear diversos tipos de unidades y formar un ejército con el que acabar con los enemigos, que a su vez crean su propio ejército y unidades propias. Para crear y mantener estos ejércitos se necesitan recursos y otros elementos que obligan a crear diversos tipos de unidades especializadas además de las militares para obtener recursos, crear edificios o curar y mantener a otro tipo de unidades.

Ilustración 21: Age of empires

Un videojuego de este tipo que tuvo bastante éxito y varias secuelas del mismo es *Age of empires* en la ilusstración 21.

2.1.10 Simuladores sociales

Este tipo de videojuegos son un subgénero de los videojuegos de simulación en el que se modela la inteligencia de diversos actores y sus relaciones. Estos videojuegos ofrecen un mundo virtual con ciertas reglas sobre el que se representan varias formas de vida autónomas con las que el jugador puede interactuar.

Algunos ejemplos de juegos de este tipo son *Los Sims*, *Singles* o *Animal Crosing*.

2.1.10.1 Los Sims

Uno de los videojuegos que inspiró este proyecto es la serie *Los Sims* de la compañía Maxis. En este videojuego catalogado como simulador social, se otorga el control de un personaje o varios dentro de un escenario en el que debe interactuar realizando las tareas cotidianas que se realizan en un hogar.

El jugador puede personalizar a su personaje y el entorno donde vive, se simulan necesidades básicas que se deben satisfacer como el hambre, la higiene y el estado emocional del personaje. El jugador debe realizar las tareas que le permitan mantener todas las necesidades de su personaje dentro de unos límites aceptables.

El juego también da la posibilidad de movimiento autónomo del personaje que actuará según el estado de sus necesidades básicas y otras características como sus gustos o el estado emocional de cada momento. En el juego también aparecen otros personajes controlados mediante las mismas técnicas de IA e interactúan con el resto de personajes del mundo virtual.

Esta serie de videojuegos ha tenido cierto éxito y han surgido varias secuelas y ampliaciones del mismo juego en forma de paquetes de mejoras. Estas mejoras incluyen numerosos objetos nuevos con los que interactuar o decorar el mundo virtual, nuevas situaciones, trabajos y escenarios en los que los personajes del juego pueden realizar diversas actividades.

Existen otros videojuegos parecidos a *Los Sims* que se diferencian de este último principalmente en el objetivo del mismo. Por ejemplo, en la serie de videojuegos *Singles* no se trata tanto de

mantener al personaje feliz generalmente y con todas sus necesidades satisfechas sino llegar a simular una relación de pareja entre dos personajes del videojuego. Para ello el jugador dispone de una serie de acciones y escenarios en los que interactúa con el otro personaje controlado de forma autónoma con técnicas de IA, interesándose por los gustos y aficiones del otro personaje.

Ilustración 22: The Sims 3

2.1.10.2 Singles

Este es un videojuego muy similar al título de Maxis *Los Sims* en referente al modelo de juego y como interactúa con el el jugador pero se diferencia con ese título en el objetivo, centrándose en las relaciones personales de compañeros de piso aunque están presentes características comunes como las necesidades básicas de los personajes y el control automático de los personajes secundarios y el personaje que controla el jugador en determinados momentos por el motor de IA.

2.1.10.3 Animal Crosing

Este es un videojuego de simulación social desarrollado por Nintendo en el que el jugador controla a un personaje de un pueblo habitado por animales antropomórficos en el que puede realizar

distintas actividades. El videojuego utiliza el reloj interno de las videoconsolas donde se ejecuta para simular el paso del tiempo de forma real por lo que algunas acciones están influenciadas por la hora real del sistema, por ejemplo el crecimiento de plantas y árboles y algunas fechas señaladas en el calendario del sistema.

2.1.11 AI-LIVE

Aunque realmente AI-LIVE no es un videojuego como tal concebido para entretener a los usuarios, sino un sistema para simular diversos entornos virtuales, actores y las relaciones entre ellos, toma elementos de varios géneros:

- Simulador social: ya que ofrece la posibilidad a un jugador de controlar a un personaje virtual e interactuar con otros actores autónomos o controlados por otro ser humano.

- Estrategia por turnos: derivada de la arquitectura cliente-servidor y basada en los turnos de los actores. Los jugadores deben elegir sus acciones en cada turno dependiendo del estado actual del mundo virtual. Si un objeto que debe utilizar para satisfacer una necesidad de su actor no es accesible en un determinado momento, puede buscar otro distinto o realizar otra acción distinta para satisfacer esa necesidad o puede planificar una serie de acciones con antelación para realizarlas en sucesivos turnos.

- AI-LIVE también tiene el componente de juego online ya que varios clientes se pueden conectar al mismo servidor a través de Internet.

2.2 Herramientas de desarrollo

En el mercado existen diferentes herramientas para la creación de videojuegos, desde simples compiladores donde hay que programar todo desde cero, algo que hoy en día casi no se utiliza ya que conlleva la programación a bajo nivel de todos los elementos gráficos del juego, desde el dibujado de un simple píxel en una pantalla compatible hasta las animaciones y efectos visuales más complejos.

2.2.1 Librerías y motores gráficos

Existen librerías gráficas que facilitan enormemente la labor del programador, facilitando las directivas básicas de dibujado,

proporcionando un motor de animaciones e incluso de partículas. Entre estas librerías se encuentran por ejemplo "*SDL*" o "*Simple Media Layer*" que es una librería de programación gráfica portada a multitud de sistemas y soportada en varios compiladores y lenguajes como *C*, *Pascal*, *Python*, *Lua* y con posibilidades para hacer uso a su vez de otras librerías como *openGL* para la renderización de gráficos en tres dimensiones.

Allegro es otra librería de distribución libre para ser usada con el lenguaje *C/C++* y permite el desarrollo de videojuegos en varias plataformas como *Windows*, *DOS*, *Unix* y otros. La librería ofrece al programador varias funciones para representar gráficos, dibujos vectoriales, sprites, funciones de matemáticas complejas y para programar y generar sonidos.

Dentro de los motores gráficos, podemos encontrar a *Irrlicht*, de código abierto y listo para usarse bajo el lenguaje *C++* y la plataforma *.Net*. También es multiplataforma ya que soporta las tecnologías *D3D* y *OpenGL* y proporciona características propias de otros motores gráficos comerciales. Existen numerosos "bindings" o enlaces a otras librerías y lenguajes de programación para extender el propio motor y ofrecer nuevas funcionalidades o para hacer uso del motor desde otros lenguajes de programación y "scripting" como *jav*a, *Perl*, *Ruby*, *Lua* o *Python*.

AI-LIVE utiliza el motor gráfico *Ogre3D* (*Object-oriented Graphics Rendering Engine 3D*). Es un motor flexible y orientado a escenas, implementado en *C++* y multiplataforma que permite crear aplicaciones gráficas usando la API de *OpenGL* o *Microsoft DirectX*.

Ogre3D es un proyecto de software libre y debido a ello ha obtenido una gran difusión y el apoyo de diversos usuarios que han colaborado en el proyecto desarrollando diversos plugins para ampliar el sistema.

2.2.2 Herramientas de creación de mundos virtuales

Kaneva es una plataforma de mundos virtuales que ofrece al desarrollador una infraestructura de servidores y herramientas de desarrollo integradas para la creación de mundos virtuales online masivos, con posibilidades para crear objetos y escenarios donde jugadores de todo el mundo pueden interactuar y donde se pueden realizar algunos negocios, generando publicidad o vendiendo añadidos, objetos o accesorios para los jugadores. Se ofrece un conjunto de herramientas diseñadas específicamente para crear videojuegos y mundos virtuales, llamada *Kaneva Game Platform* e

incluye el cliente para conectarse o lanzar la aplicación en sí, los distintos componentes de desarrollo, un sistema de inteligencia artificial con el que controlar agentes autónomos en el mundo virtual y los servidores necesarios para toda la infraestructura.

La idea de *Kaneva* es que los estudios desarrolladores de videojuegos puedan lanzar juegos de este tipo en mucho menos tiempo, con menor coste y centrándose solamente en la creación de una historia y añadido de componentes y objetos en lugar de crear todos los componentes necesarios y ensamblarlos. Este tipo de videojuegos necesitan ajustar el uso de los objetos que pueden realizar los actores y los efectos que provocan en ellos y el mundo virtual. En general suele haber una gran cantidad de objetos y obtener un sistema equilibrado conlleva mucho tiempo y pruebas a realizar con distintos valores. *Kaneva* ofrece un sistema básico para añadir objetos y controlar los efectos que producen al usarlos, ofrece también un sistema para modelar objetos, un sistema básico para programar la IA de los actores y otras herramientas.

Kaneva es extensible mediante los lenguajes de scripting *Lua* y *Python*, ofreciendo una gran flexibilidad para crear modos de juego y añadir nuevas funcionalidades, además de permitir programar la inteligencia artificial de los actores.

Otra plataforma similar a *Kaneva* es *Multiverse*, que ofrece unas herramientas muy parecidas de desarrollo para generar diversos mundos virtuales, herramientas para explorar estos mundos y la posibilidad de generar negocio dentro de los mundos desarrollados con la herramienta. A diferencia de *Kaneva*, la infraestructura de red y los servidores necesarios para alojar el mundo virtual son responsabilidad del desarrollador y se ofrece la posibilidad de crear mundos virtuales libres, sin costo alguno para el desarrollador, siempre que este tampoco imponga ningún coste a los usuarios del mundo virtual generado con la plataforma.

Second Life es otro mundo virtual en el que los jugadores se conectan controlando a un personaje o avatar creado por ellos mismos o seleccionado de entre varios posibles. Los jugadores pueden interactuar con otros jugadores mediante sus avatares. Las acciones que pueden realizar van desde comunicarse con otro jugador o grupo de jugadores a través de un chat, escrito o por voz, intercambiar objetos o realizar diversas actividades utilizando los distintos objetos del mundo virtual. Existen multitud de actividades que los jugadores pueden realizar ya que el sistema ofrece la posibilidad de añadir nuevos objetos al mundo virtual y mediante su

propio lenguaje de script, el Linden Scripting Language que permite proporcionar una funcionalidad a los objetos o edificios introducidos.

Los avatares no tienen ninguna necesidad básica que cumplir y por lo tanto no tienen tampoco la necesidad de comprar objetos para continuar jugando pero en el mundo virtual de *Second Life* existe una economía propia, basada en una moneda virtual llamada *dólar Linden*. Los jugadores pueden adquirir ese dinero virtual al realizar algunas acciones en el juego o proporcionando un servicio que otro jugador requiere. Los jugadores también pueden comerciar con sus objetos o pueden comprar esta moneda virtual invirtiendo dinero real. La tasa de cambio fluctúa diariamente y se realiza a través de la aplicación cliente que utilizan los jugadores para conectarse. El dinero va a parar sirve para financiar a la propia *Second Life*.

Empresas externas y usuarios crean diverso contenido para el juego con distintos fines, como una forma más de publicitarse en la red o por mera diversión. El contenido creado puede distribuirse de forma libre o gratis o usando la moneda virtual del juego.

En el mundo de *Second Life* no existen personajes propiamente controlados por una IA aunque hay distintas aproximaciones en las que los usuarios crean scripts para objetos que reaccionan ante distintos escenarios como la acción de un jugador sobre ellos o que ofrecen algún tipo de respuesta ante una acción suya u objeto que lleve con él. Hay otras aproximaciones que intentan programar una IA compleja sobre este mundo virtual, creando los llamados NPC (de sus siglas en inglés Non-Player Character, o personaje no controlado por un jugador) como los agentes de inteligencia virtual creados por Novamente. Entre ellos se encuentran varios animales que actúan como mascotas en el mundo virtual de *Second Life* como un loro o un perro y cuyo comportamiento no está predefinido sino que mediante el uso de distintas técnicas como algoritmos genéticos y compartición de conocimiento entre varios agentes que crean así una inteligencia colectiva, se va enriqueciendo conforme se interactúa con los agentes, aprendiendo distintas acciones o contestando a sencillas preguntas.

2.3 Arquitecturas de red

Las redes de ordenadores se pueden clasificar por la relación funcional entre los elementos de la red. Según esta clasificación, existen las redes con arquitectura cliente-servidor o peer-to-peer.

- Cliente-servidor: es la arquitectura de red clásica centralizada donde un equipo hace de servidor y controla

todas las comunicaciones entre los distintos clientes, pasando todas ellas por este servidor central.

- Peer-to-Peer: es una arquitectura de red descentralizada en la que todos los nodos comparten sus recursos, intercambiándolos con otros nodos de forma equitativa.

2.3.1 Arquitectura Cliente-Servidor

Esta arquitectura se basa en dos roles, el rol de servidor y el rol de cliente. Los nodos clientes inicial la comunicación con el servidor enviando peticiones y el servidor responde a esas peticiones. Un ejemplo claro es Internet y los navegadores web que hacen de clientes en este caso, realizando peticiones a los servidores que tienen alojados los contenidos a los que se quiere acceder.

Existen multitud de servidores distintos para diferentes aplicaciones como pueden ser servidores ftp para intercambio de archivos, servidores de bases de datos, servidores de nombres de dominio o servidores de correo.

En algunas ocasiones un servidor puede hacer algunas funciones de los clientes. Por ejemplo en una red de servidores, todos conectados entre sí, deben redirigir las peticiones de los clientes hacia otros servidores, realizando así una consulta a otro servidor y ejerciendo de esta forma el rol de cliente.

Generalmente los servidores son aplicaciones en continua ejecución que atienden un gran número de peticiones y suelen ejecutarse sobre hardware dedicado a ese fin aunque también pueden ejecutarse en los mismos equipos que los clientes.

2.3.2 Arquitectura Peer-to-Peer

En este tipo de arquitectura todos los nodos son equivalentes, no hay ningún nodo central que sincronice al resto de nodos.

Cada nodo o *peer* proporciona una proporción de sus propios recursos como tiempo de proceso de la CPU, espacio de almacenamiento o ancho de banda a la red P2P. Cada nodo proporciona recursos a la vez que los obtiene de la red.

Este tipo de arquitectura se utiliza especialmente en el intercambio de ficheros en red distribuyendo la carga de trabajo y

ancho de banda necesario para transmitir todos los datos entre todos los nodos de la red.

2.4 Técnicas de IA usadas

Uno de los temas básicos en la inteligencia artificial es la resolución de problemas y se emplean distintos algoritmos que ayudan a abstraer el problema a resolver, identificar las acciones que se deben tomar para llegar a una solución y resolverlo de la forma más eficiente, o encontrar todas las soluciones posibles al problema.

En el siguiente punto se describen algunas de estas técnicas.

2.4.1 Sistemas de producción

A la hora de resolver un problema mediante el uso de sistemas de computación, no siempre se puede definir el problema bajo unas premisas matemáticas estrictas y formular una solución aplicando diversos cálculos intermedios hasta llegar a la solución. A veces se buscan todas las soluciones de un problema, la solución óptima o simplemente hay alguna restricción para un problema o un paso intermedio en la solución del mismo que impide resolverlo mediante las técnicas de programación clásicas.

Existe otra forma de afrontar este tipo de problemas o como alternativa a un programa clásico y es el uso de sistemas basados en el conocimiento del problema, en el que se recoge toda la información del problema a resolver, se crean unas reglas básicas con las que proceder y se toman decisiones sucesivamente hasta dar con la solución. Estos sistemas expertos basados en reglas se definen con un conjunto de módulos o partes que lo caracterizan:

- **Base de conocimiento**:contiene la definición del problema a resolver de una forma estructurada, generalmente sencilla y basada en reglas con un antecedente y un consecuente.

- **Motor de inferencia**: es el módulo que se encarga del proceso de razonamiento, buscando y aplicando las reglas de una forma ordenada.

- **Memoria de trabajo**: contiene la información que permanece invariable en el problema, los hechos que son verdaderos, datos de entrada y conclusiones intermedias que se generan al aplicar las distintas reglas mediante el proceso de razonamiento del motor de inferencia.

- **Interfaz de usuario**: módulo que permite la comunicación entre el usuario y el sistema experto.

- **Generador de explicaciones**: es un módulo opcional y ofrece un resumen de la estrategia para resolver el problema que se ha usado y una traza con las decisiones tomadas.

- **Sistema de adquisición de conocimiento**: permite incluir nuevo conocimiento o actualizar el existente.

Las reglas son proposiciones lógicas que relacionan dos o más objetos y se divide en dos partes, la premisa y la conclusión. La premisa y la conclusión están formadas a su vez por expresiones lógicas del estilo objeto-valor y conectadas mediante operadores lógicos. Si se cumplen todas las condiciones de la premisa, entonces se realizan las acciones que marca la conclusión.

Las reglas y su ejecución es la base en el desarrollo de AI-LIVE ya que tanto el servidor como los clientes manual y de IA autónoma están programados mediante reglas en un sistema experto llamado *CLIPS*.

2.4.1.1 Clips

CLIPS (*C Language Integrated Production System*) fue creada por la Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center en 1984. Se desarrolló con el objetivo de facilitar el desarrollo de sistemas expertos. Un sistema experto es una aplicación informática diseñada para resolver diversos problemas que exigen un gran conocimiento sobre un determinado tema. El sistema experto actúa sobre una base de conocimiento e imita las actividades que normalmente realiza un humano para resolver un problema en cuestión.

CLIPS se diseñó siguiendo los objetivos de crear un software con alta portabilidad, bajo coste y facilidad de integración. Está desarrollado en *C* y permite una integración completa con *C* y otros programas como Ada y Fortran. De esta forma se puede hacer uso de rutinas de *CLIPS* dentro del código de otro programa y después *CLIPS* devuelve el control al sistema que hace uso de sus funciones. *CLIPS* también permite la extensión del propio lenguaje, proporcionando una interfaz para crear nuevas funciones externas y después poder hacer uso de ellas dentro de un programa escrito en *CLIPS*.

En el proyecto se ha hecho uso de las dos opciones, haciendo uso de las funciones y procedimientos de *CLIPS* dentro del código del servidor y clientes y añadiendo funciones externas a *CLIPS*, para agilizar los cálculos realizados en el motor emocional usado para simular la psique de los actores.

CLIPS ofrece paradigmas heurísticos (reglas) y procedimentales (funciones y objetos) para representar el conocimiento.

Las reglas se utilizan para representar heurísticos que especifican un conjunto de acciones a realizar para una situación dada. Al diseñar un sistema experto en *CLIPS*, se define un conjunto de reglas que aplicadas con una determinada lógica sobre un conjunto de hechos o conocimiento dados, resuelven un problema. En el videojuego AI-LIVE, todas las acciones y decisiones que toman los actores están especificadas mediante un conjunto de reglas de forma que si se cumplen ciertas condiciones, el actor realizará una acción u otra.

2.4.2 Planificación automática

Los planificadores en inteligencia artificial tratan de resolver un problema dado partiendo de un estado inicial y utilizando diversos operadores para conseguir unas metas dadas.

Los planificadores toman como entradas un problema y un dominio:

1. El problema es la descripción del estado inicial y las metas que se persiguen, utilizando predicados lógicos.

2. El dominio consiste en un conjunto de operadores que permiten la transición de un estado lógico a otro.

3. Los operadores se describen utilizando precondiciones y los efectos que produce la aplicación de este operador.

Existe el llamado espacio de estados que es el conjunto de todos los estados posibles del problema. Este conjunto suele ser muy grande y para encontrar un plan que solucione el problema, se realiza una búsqueda en el espacio de estados.

2.4.2.1 Prodigy

Prodigy es el planificador utilizado como cliente de IA en el proyecto AI-LIVE. Es un planificador de tareas independiente del dominio capaz de incorporar conocimiento de control para guiar la búsqueda.

A *Prodigy* se le proporciona el dominio, que son los operadores y reglas de inferencia: el problema y conocimiento de control en forma de reglas "if then" y *Prodigy* devuelve el plan que satisface las metas planteadas del problema.

2.5 Lenguajes de script

Los lenguajes de script son lenguajes de programación diseñados originalmente para controlar otros sistemas y se diferencian principalmente de los lenguajes de programación tradicionales en que no son compilados sino interpretados. Estos lenguajes suelen ser multiplataforma ya que se programa un intérprete que es portado a las distintas arquitecturas y después se escriben los llamados scripts que son instrucciones para el intérprete y generalmente pueden ejecutarse en distintos sistemas operativos sin alterar nada del código escrito.

Estos lenguajes se utilizan mucho para automatizar distintas tareas o simplificar ciertos trabajos complejos que requieren varias pasos como tareas de backup, escaneando distintos directorios, seleccionando los archivos a guardar, después comprimir estos archivos en un determinado formato y por último enviar estos backup a otro lugar donde serán tratados o guardados.

Con el tiempo estos lenguajes han ido evolucionando y creciendo en popularidad, convirtiéndose en lenguajes de uso general, creando diversas librerías y módulos que amplían las capacidades de estos lenguajes de forma que ya no son simplemente un conjunto de instrucciones que un intérprete ejecuta, sino que se realizan programas completos siguiendo los mismos paradigmas de programación que los lenguajes más tradicionales.

Existen multitud de lenguajes de script distintos y se usan también en entornos muy diversos como en la navegación web, para extender la funcionalidad de otros programas o videojuegos o procesadores de texto. De entre todos ellos hablaremos del lenguaje *shell de Bourne*, *PHP* y P*ython*.

2.5.1 Shell script

Una shell es una pieza de software que proporciona una interfaz a los usuarios de un sistema operativo y proporciona acceso a los servicios del mismo kernel del sistema operativo que es el núcleo del sistema, el que gestiona a bajo nivel todos los componentes del sistema. Hay multitud de shells disponibles para los distintos sistemas operativos, basadas en texto como:

1. Shells basadas en texto
   - Shells Unix como *Bourne Shell* (*sh*), *Korn shell* (*ksh*), *rc shell* (*rc*) o *remote shell* (*rsh*)
   - Shells No-Unix como *cmd.exe*, *Amiga Cli/AmigaShell* o *Windows PowerShell*.
2. Shells gráficas como *Windows Shell*, *Fluxbox*, *Enlightenment*, *Gnome* o *Xfce*.

Los shell scripts son archivos ejecutables escritos en el lenguaje y con la sintaxis de la shell y contienen una serie de comandos, variables y sentencias de control de flujo como bucles y sentencias condicionales.

En el proyecto se ha utilizado para automatizar la ejecución y facilitar al usuario final ejecutar los distintos módulos, lanzando varios clientes, el servidor o la interfaz gráfica.

2.5.2 PHP

PHP o *Hypertext Preprocessor* se diseñó originalmente para el desarrollo web ofreciendo contenido dinámico. El código *php* se añade dentro del archivo *HTML* y se interpreta en el servidor web, generando el documento web de forma dinámica. Este lenguaje de scripting se puede interpretar mediante línea de comandos con una aplicación específica y es capaz de realizar tareas en el sistema operativo como cualquier otro lenguaje de programación.

En la versión anterior del proyecto se utilizaba para lanzar los distintos módulos como el servidor o un cliente de forma separada. Cada módulo tenía su propio script en *php* que servía para ejecutarlo, usando unos archivos de configuración.

2.5.3 Python

Python es un lenguaje interpretado de alto nivel de uso general que está diseñado para que su sintaxis sea muy legible. Tiene una

gran librería con muchas funciones ya implementadas y debido a su popularidad, muchos desarrolladores han creado diversas bibliotecas para ampliar el lenguaje.

Los programas escritos en *Python* se pueden distribuir como un script escrito en *Python* que es ejecutable mediante un intérprete de *Python* o como un ejecutable que no necesita de otra herramienta para funcionar.

Python es un lenguaje multi-paradigma, soportando la programación orientada a objetos, programación estructurada y algunos aspectos de la programación funcional y orientada a aspectos (incluyendo metaprogramación). Se ha utilizado como lenguaje de script en muchas aplicaciones, para ampliar su funcionalidad sin necesidad de recompilar como *Blender*, *GIMP*, *Inkscape* o aplicaciones puramente escritas en *Python*, usando librerías portadas a este lenguaje (*PyGame*, *PyGTK*, etc.) como el juego *Frets On Fire*.

2.6 Herramientas de compilación

2.6.1 Make

Make es una herramienta para compilar programas y librerías de forma automática desde su código fuente, utilizando unos archivos llamados "Makefiles" que especifican las dependencias entre los archivos fuente y cómo obtener el programa final. *Make* se puede utilizar para detectar cambios en un archivo de imagen y ejecutar una serie de acciones para transformar la imagen a otro formato, copiar la imagen resultante a otro destino e informar a una lista de usuarios por e-mail con las acciones que se han llevado a cabo.

En los Makefiles se definen las dependencias entre archivos y las acciones que se deben llevar a cabo para transformarlos y otras reglas para instalar el producto obtenido o eliminarlo del sistema.

2.6.2 Scons

Scons es una herramienta para compilar software basada en el lenguaje *Python*. Se puede distribuir junto a la herramienta desarrollada y los scripts necesarios en *Python* para compilarla y de esta forma producir en la plataforma objetivo los ejecutables de la aplicación, no se necesita ninguna otra herramienta para compilar, ni si quiera es necesario que el usuario final tenga instalado Python para compilar, se puede usar el ejecutable distribuido directamente.

Scons incorpora un análisis automático de dependencias para los lenguajes *C*, *C++* y *Fortran* y se puede extender integrando otras herramientas para diversos lenguajes.

Scons basa su funcionamiento en un script escrito en *Python* que es en sí mismo un programa para compilar el proyecto, declarando variables, dependencias y el compilador a usar. Ejecutando este script se obtienen los binarios finales del programa desarrollado.

2.6.3 Cmake

CMake es un sistema para construir software, multiplataforma e independiente del compilador, funcionando con los clásicos *Make*, *Xcode* y *Microsoft Visual Studio*. Está desarrollado por la empresa *Kitware* y es de código libre.

CMake se utiliza para controlar el proceso de compilación utilizando archivos de configuración simples independientes de la plataforma y compilador final usado para construir el software. *Cmake* genera archivos "Makefiles" nativos para la arquitectura elegida. Puede crear los ejecutables en el mismo directorio que los archivos fuente o en otro completamente distinto, soporta la creación de distintas librerías y ejecutables desde la misma estructura de archivos fuentes. Soporta complejas jerarquías de directorios con distintos módulos y librerías y situaciones en las que se deben crear ejecutables que luego serán enlazados para construir otra aplicación.

El proceso de compilación se controla creando uno o más archivos de configuración llamados CmakeLists.txt en cada directorio, incluidos todos los subdirectorios del proyecto que lo conforman. Cada uno de estos archivos consiste en una serie de comandos y cada comando puede tener distintos argumentos. *CMake* incluye una serie de comandos predefinidos y ofrece la posibilidad de crear otros nuevos, mediante el propio lenguaje, creando funciones o incluyendo otros generadores automáticos de Makefiles. Con estos simples archivos de reglas, se pueden generar los distintos Makefiles para *Unix*, *Windows*, *Mac OS X*, *MSVC*, *Cygwin*, *MinGW* y *Xcode*.

Grandes proyectos como el desarrollo de *KDE* u *Ogre* han adoptado esta herramienta para facilitar del desarrollo y compilación en distintas arquitecturas al igual que otros proyectos de diversa índole como *Avidemux* para el procesamiento de vídeo, *MySQL* en el campo de las bases de datos o *iCub* en el campo de la robótica.

2.7 Control de versiones

El software de control de versiones se utiliza principalmente en el desarrollo de software para controlar los diversos cambios que se producen en los distintos ficheros de un proyecto, desde los archivos con el código fuente a la documentación. Estos programas ayudan a registrar los diversos cambios que un equipo de trabajo realiza sobre un proyecto, haciendo posible volver a versiones anteriores o incluso dividir el proyecto en otro completamente distinto, con otros desarrolladores involucrados.

Estos sistemas cuentan con herramientas que aseguran la consistencia de los archivos, proporcionando métodos para bloquear un archivo e impedir que diversos cambios producidos por distintos desarrolladores entren en conflicto y se puedan perder mientras se guardan. Existen sistemas centralizados basados en la arquitectura cliente-servidor donde el servidor mantiene las distintas versiones de los archivos y sistemas distribuidos basados en una red p2p donde cada cliente guarda una copia del proyecto y se sincronizan mediante ficheros de cambios entre los clientes.

2.7.1 CVS

Las características de este sistema de versiones son:

- Versión individual de los ficheros: cada fichero tiene su número de versión y los cambios realizados varían este número de versión pero solamente al fichero cambiado.

- Envío de ficheros completos: cuando se realiza un cambio en un fichero, se sube el fichero completo al servidor, no solamente los cambios.

- No permite renombrar o borrar ficheros de forma automática, hay que hacerlo manualmente, creando un fichero nuevo y borrando el antiguo.

- No se versionan los enlaces o accesos directos a ficheros para evitar un posible fallo de seguridad. Un usuario podría crear un enlace al fichero /etc/passwd con el nombre "index.html" por ejemplo y al publicar la revisión del código con cvs en un servidor web, quedaría expuesto al público el contenido de un fichero de seguridad del sistema operativo.

2.7.2 SVN

Las características de este sistema de versiones son:

- "Commits" son operaciones atómicas: los "commits" son las actualizaciones que realiza un desarrollador, enviando los cambios realizados al servidor que mantiene el repositorio. Se dice que es una operación atómica porque si alguno de los cambios falla al enviarse al servidor, se cancela la operación y la versión del código que había en el repositorio permanece sin cambios.

- Soporte para renombrar, mover o borrar archivos y directorios. Aunque en los clientes se realiza la operación con una sola instrucción, en el servidor realmente borra el archivo antiguo y se crea uno nuevo con el nombre o ruta cambiados. Esta operación es transparente para el usuario.

- Se mantienen los cambios en la estructura de directorios, cambios de nombres en archivos y directorios y las propiedades de los propios archivos como permisos de lectura, escritura y ejecución aunque no se registran los cambios en la fecha de acceso, creación o modificación. Se pueden copiar y mover directorios completos y mantener un histórico de todos los cambios hechos.

- Se versionan los enlaces o accesos directos a directorios o archivos.

- Se versiona todo el proyecto, no los archivos individualmente por lo que al confirmar los cambios a un conjunto de archivos, se aumenta el número de versión del proyecto y el directorio que contiene a los ficheros modificados.

- Los cambios se notifican con ficheros de cambios. Esto es que si solamente se modifican unas líneas de un fichero fuente muy grande, el cliente y el servidor solamente intercambiarán un fichero que identifica las líneas cambiadas, no todo el fichero fuente cambiado, ahorrando ancho de banda utilizado y agilizando el sistema haciéndolo más rápido que otras alternativas.

2.7.3 Bazaar

Bazaar es un sistema de control de versiones muy versátil, puede actuar como un sistema centralizado o distribuido y permite al desarrollador trabajar de forma local con un repositorio de software y llevar un control de sus cambios locales en el código además de colaborar con otro equipo de desarrollo en un proyecto donde se lleve un control de versiones centralizado o incluso mantener los dos sistemas, centralizado y distribuido en el mismo proyecto. Las principales características de este sistema son:

- Alta eficiencia y rapidez en todas las acciones que se pueden realizar.

- Posibilidad de trabajar offline.

- Facilidad para integrarse con proyectos ya gestionados por otras herramientas de control de versiones.

- Es multiplataforma: tanto la interfaz gráfica de usuario como por línea de comandos funcionan en distintos sistemas operativos sin problemas.

- Soporte para renombrar y mover archivos y directorios fácilmente. Muchos otros sistemas de control de versiones tienen problemas si dos o más desarrolladores cambian la estructura de directorios de forma simultánea y añaden o borran ficheros en la nueva estructura de directorios. Al aplicar los cambios, no todas las herramientas obtienen el resultado esperado. Bazaar trata las operaciones de renombrado de forma distinta, evitando cualquier tipo de problemas.

2.8 Tecnologías utilizadas

En este apartado se mencionan las tecnologías que se han utilizado en el proyecto y el por qué se han utilizado frente a otras opciones.

- **Arquitectura cliente-servidor**

Esta es una arquitectura clásica que utilizan numerosas aplicaciones en la que intervienen dos roles con funcionalidades diferenciadas.

- Cliente: la funcionalidad del cliente es representar la información y hacer de interfaz de usuario para la aplicación. El cliente realiza peticiones al servidor.
- Servidor: el servidor recibe las peticiones de los clientes y les envía las respuestas a esas peticiones.

La arquitectura cliente-servidor es un sistema centralizado donde el servidor recibe peticiones de uno o más clientes y ofrece un servicio determinado a estos clientes como intercambio de ficheros, mensajes u otros datos.

Esta arquitectura es la que estaba implementada anteriormente y ha demostrado ser bastante eficiente para el proyecto AI-LIVE, ofreciendo un control sobre el universo virtual y la sincronización que necesitan los distintos clientes.

- **Sistema Make para compilar el código**

Se decide mantener el sistema de compilación pero proporcionando unos nuevos ficheros Makefile más optimizados y multiplataforma. Se han reescrito algunas funciones del código fuente para hacerlas compatibles con otros sistemas y se ha proporcionado un fichero Makefile con el que compilar tanto en entornos *Linux* como *Windows*.

Se ha mantenido este sistema ya que es bastante sencillo de mantener y el script creado para compilar no es excesivamente complejo, eliminando la necesidad de utilizar las herramientas para generar un script Makefile en cada plataforma. El Makefile creado es compatible para entornos *Linux* y *Windows*.

- **Sistema de arranque en bash**.

Se ha optado por un script escrito en bash para Linux para facilitar la ejecución de los distintos módulos. En la versión anterior del proyecto se utilizaba un script en *PHP* para cada módulo y ahora con un solo script escrito en el lenguaje con el que cuentan la mayoría de distribuciones *Linux* se pueden ejecutar los distintos módulos de una manera más cómoda y efectiva.

Se ha implementado este sistema para eliminar la necesidad de instalación de los componentes *PHP* y a la vez hacer configurable el entorno de ejecución.

- **Control de versiones**

La adopción de un sistema de control de versiones como es *SVN* ha facilitado el desarrollo del proyecto. En la versión anterior del proyecto los distintos desarrolladores de los módulos debían incorporar en su código los cambios que realizaban los demás a mano, siendo a veces un trabajo muy costoso ya que por ejemplo, una acción implementada en el cliente, debe estar también implementada en el servidor para poder actualizar el estado del escenario virtual y que estos cambios lleguen después al resto de clientes.

Se ha optado por el sistema *SVN* alojado en el servicio online *Google Code* ya que simplifica la administración del servidor, ofreciendo además diversas herramientas como una propia *Wikipedia* y un sistema para la descarga de ficheros.

3 OBJETIVOS DEL PROYECTO FIN DE CARRERA

El objetivo del presente proyecto de fin de carrera es proporcionar nueva funcionalidad a la aplicación AI-LIVE, creando nuevas necesidades básicas para los actores y en general mejorar la aplicación para dotarla de más realismo.

Anteriormente el número máximo de actores conectados dependía del escenario creado porque se utilizaba una bolsa con objetos y los actores interactuaban con ellos. De esta forma se limitaba el número de actores al número de bolsas que había en el escenario. En este proyecto se ha modificado el universo virtual, cambiando todas las acciones de los actores para permitirles actuar con cualquier objeto del escenario y poder tener ciertos objetos en su poder para usarlos en un momento dado. De esta forma se ha incrementado el número de actores de forma que sólo estará limitado por el tamaño del escenario y la cantidad de objetos que hay en él disponibles para que los actores puedan interactuar.

Introducir el concepto de tiempo en el universo AI-LIVE. A pesar de que la aplicación está basada en el modelo cliente-servidor otorgando turnos a cada cliente, se puede introducir el concepto de tiempo añadiendo una duración determinada a ciertas acciones como dormir o trabajar, controlando la duración en turnos de forma que se puede contabilizar el paso del tiempo aunque haya actores controlados por un cliente manual cuyo tiempo de respuesta es indefinido, puede tardar unos segundos en elegir la acción a realizar o varios minutos.

Otro objetivo del proyecto es integrar el motor emocional desarrollado en otro proyecto de fin de carrera de Marta Jiménez Matarranz. Esta integración se ha realizado integrando el sistema de gustos con las acciones de los clientes y añadiendo algunas reglas complementarias para permitir actuar a los actores según sus gustos y preferencias.

La aplicación se diseñó en principio para ser portada a otros sistemas como Windows pero había que adaptar el código y encontrar un compilador compatible. Se ha creado un archivo para permitir la compilación del proyecto tanto en plataformas Unix como Windows.

Actualizar la versión de *CLIPS* usada en el código fuente a la última versión disponible. Desde que se inició el proyecto han

aparecido nuevas versiones del código fuente de CLIPS y no se había actualizado.

Mejorar y adaptar el sistema de compilación. Se trata de eliminar la recursividad a la hora de compilar y mejorar el tiempo de compilación de forma general.

Introducir nuevas necesidades básicas y modificar la evolución de las ya creadas. Se trata de ampliar el proyecto con más posibilidades y crear las acciones y objetos necesarios para controlar esas necesidades. Las necesidades añadidas son "solitude" o soledad y "thirst" que representa la sed del actor. La soledad se utiliza para controlar la necesidad del actor para comunicarse y relacionarse con otros actores. La sed permite usar e introducir otros objetos en el escenario.

Centralización de los perfiles para los actores. Los perfiles describen las características de los actores. Se pueden definir los gustos de los actores y diversos parámetros como el nombre, gráfico que se emplea en el GUI, sexo del actor, capacidad para cargar con objetos o los niveles de los drives o indicadores que al actor debe satisfacer, por ejemplo, con el drive "maxhunger" que mide el hambre del actor, se define a partir de qué valor el actor siente hambre y debe buscar algún objeto de comida en el escenario. En la versión anterior del proyecto, cada cliente tenía definidos sus propios perfiles de clientes, ahora todos los perfiles se encuentran en un mismo sitio y son accesibles por todos los clientes.

Facilitar la ejecución de los distintos módulos. Se ha mejorado el sistema de ejecución, cambiando los distintos scripts que tenía cada módulo para ejecutarse y creando un solo script para Unix capaz de ejecutar los módulos del proyecto según los parámetros que se definan. Se pueden lanzar varios clientes controlados por la IA y otros tantos manuales, además del servidor o el cliente gráfico con un solo script creado y su archivo de configuración correspondiente.

3.1 Estado anterior

En un principio el proyecto empezó con la implementación del servidor, un cliente de inteligencia artificial utilizando el sistema experto *CLIPS*, un cliente de inteligencia artificial basado en el planificador *Prodigy* y un cliente GUI gráfico con un motor en dos dimensiones escrito en *C* y utilizando las librerías *SDL* llamado *CREND*.

El proyecto ha sido ampliado varias veces por distintos alumnos, añadiendo un sistema de necesidades básicas, un motor emocional y la evolución de una interfaz gráfica tridimensional.

Ilustración 23: Primer GUI: CREND

Ilustración 24: Cliente GUI 3d vertical

Ilustración 25: Cliente GUI 3d ángulo

En la versión anterior del proyecto se encontraban separados el sistema de necesidades básicas y el motor emocional. Había programado un cliente de acceso manual y un generador de escenarios.

Los archivos de configuración de los actores o perfiles, estaban guardados en varios puntos dentro del árbol de directorios de la aplicación, existiendo el mismo perfil con la misma configuración en distintos módulos. Si se quería utilizar el mismo perfil determinado en el cliente de IA y un cliente manual para comparar el comportamiento por ejemplo, debía copiarse su archivo de configuración en los dos módulos, el directorio del cliente de IA y el del cliente de acceso manual. Además cualquier cambio que se hiciera en estos perfiles, debía replicarse en todos los archivos y la carga de estos archivos de configuración estaba fijada en un determinado directorio, no se podía cambiar de localización.

Para compilar el proyecto existía un script Makefile para cada módulo y se debía compilar desde el propio directorio de cada módulo ya que se usaban rutas relativas en los distintos archivos fuentes de la aplicación.

Para ejecutar cada módulo también había un script por cada uno de ellos y por ejemplo un cambio en el puerto de escucha del servidor obligaba a cambiar ese mismo puerto en todos los scripts.

4 GESTIÓN DEL PROYECTO

A continuación se detallan las tareas en las que se ha dividido el trabajo realizado, indicando el objetivo de cada tarea. Se indicará el tiempo empleado en cada tarea y el gasto económico derivado de cada tarea.

Se incluye el diagrama de Gantt para el proyecto y un presupuesto con el coste de los recursos empleados y el coste final del proyecto.

4.1 Descomposición en tareas

El proyecto se ha dividido en varias tareas y cada tarea está formada por un conjunto de estudios y actividades que se han llevado a cabo.

4.1.1 Actividad A: análisis del problema

Esta tarea consiste en analizar el problema a resolver, estudiar la arquitectura anterior y plantear una solución a los objetivos marcados.

4.1.2 Actividad B: documentación y análisis del estado del arte

En esta fase de investigación, se busca información del estado actual de las tecnologías empleadas en este proyecto y se ofrece una visión general de las mismas, junto con alternativas disponibles en el mercado.

- **Tarea B1**: investigación teórica de las tecnologías
  Se analizan las distintas tecnologías empleadas en el proyecto y las necesidades para conseguir los objetivos marcados. Se busca información sobre tecnologías alternativas.

- **Tarea B2**: estudio de viabilidad
  Se estudia si los objetivos planteados son alcanzables utilizando las tecnologías seleccionadas y adaptando la arquitectura actual a los nuevos requisitos. Se seleccionan las herramientas necesarias para el desarrollo del proyecto.

4.1.3 Actividad C: diseño de la aplicación

En esta fase se han diseñado un plan de acción para modificar la arquitectura actual de AI-LIVE para adaptarla a los nuevos requisitos, estudiando los módulos de la aplicación a modificar y cuáles se necesitan añadir para cumplir los objetivos marcados.

4.1.4 Actividad D: implementación de la solución

Tras la fase de diseño donde se determinan los puntos a mejorar de la arquitectura anterior de AI-LIVE, se procede a la fase de implementación. Se utilizan las herramientas informáticas adecuadas decididas en el estudio de viabilidad.

4.1.5 Actividad E: evaluación de la aplicación

En esta fase, se realizan las pruebas con diferentes agentes y se controla la evolución de las necesidades básicas de los actores que controlan, comprobando que se mantienen dentro de unos valores lógicos y comprobando que realizan todas las acciones disponibles según unos parámetros.

- **Tarea E1**: definición de las pruebas
  Se definen qué parámetros se van a estudiar y se fijan unos valores determinados para los distintos actores controlados por los agentes de IA.

- **Tarea E2**: realización de las pruebas
  Se ejecuta la aplicación con los parámetros fijados y se generan los datos sobre los actores para su posterior análisis.

- **Tarea E3**: análisis de los resultados
  Se analizan los resultados obtenidos y se ofrecen unas conclusiones acerca de los mismos.

4.1.6 Actividad F: redacción de la memoria

Esta tarea es el proceso en el que se redacta la presente memoria y se realiza durante toda la duración del proyecto.

4.2 Duración de las tareas

A continuación se ofrece una tabla con la duración de las tareas especificadas en el apartado anterior y el coste. Se han realizado los cálculos del coste con un precio de 40€/hora por el trabajo de un ingeniero en el proyecto.

B1. Investigación teórica de las tecnologías	30	4	120	4800
B2. Estudio de viabilidad	28	4	112	4480
C. Diseño de la aplicación	28	4	112	4480
D. Implementación de la aplicación	40	3	120	4800
E. Evaluación de la aplicación				
E1. Definición de las pruebas	4	2	8	320
E2. Realización de las pruebas	2	3	6	240
E3. Análisis de los resultados	4	2	8	320
F. Redacción de la memoria	120	1	120	4800
		Total proyecto	614	24560

Tabla 1: Duración de tareas y coste

El coste total del proyecto asciende a 24560 euros.

4.3 Diagrama de Gantt

En la siguiente ilustración número 17 se muestra el diagrama de Gantt del proyecto con todas tareas mencionadas anteriormente.

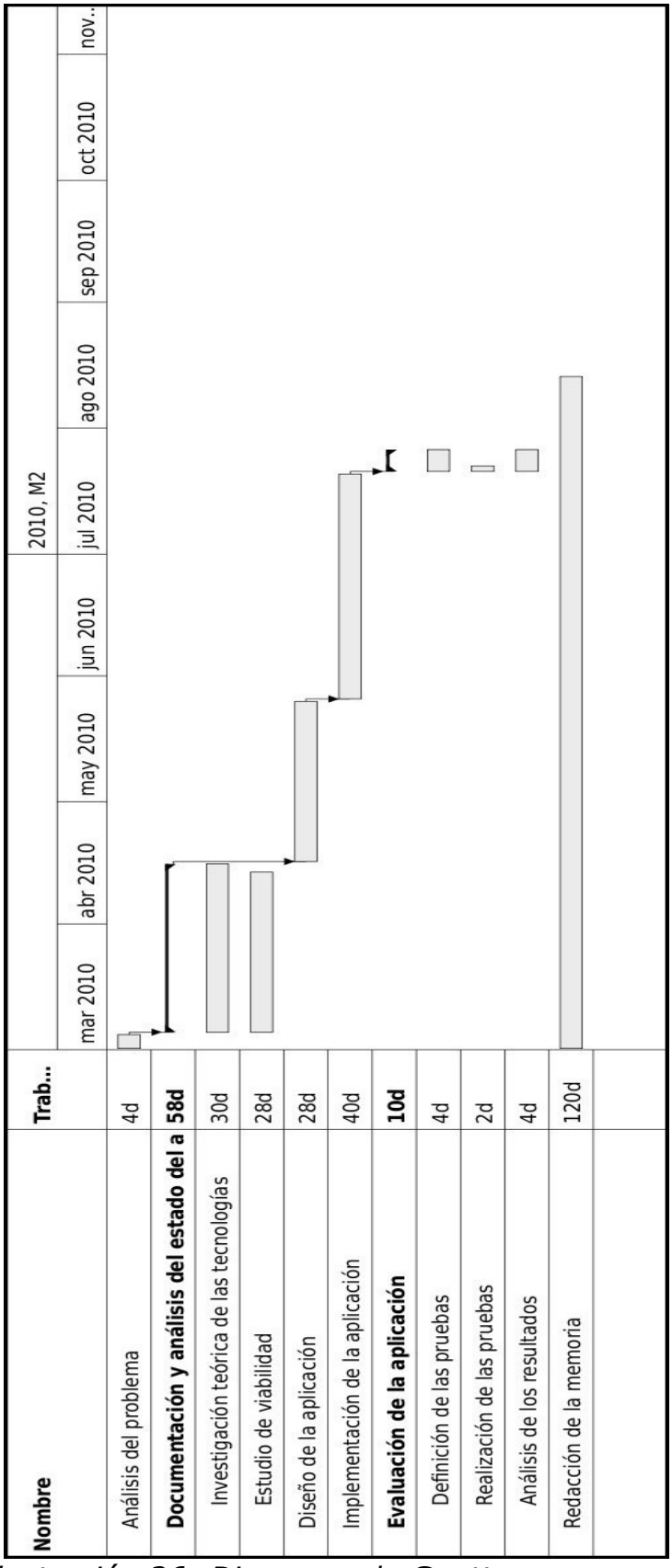


Ilustración 26: Diagrama de Gantt

5 MEMORIA-TRABAJO REALIZADO

A continuación se describe el trabajo realizado, ofreciendo una visión general del proyecto y después describiendo los cambios en los módulos realizados.

5.1 Introducción

En este apartado de la memoria se describe el trabajo realizado para la implementación del servidor, repasando el estado del sistema AI-LIVE y detallando las modificaciones y los avances introducidos.

Posteriormente se explicarán los módulos de la arquitectura utilizada que componen la aplicación AI-LIVE, así como la comunicación entre ellos, las entradas y salidas que generan; y el modelo de conocimiento de la aplicación, que contiene el diagrama de clases con sus atributos y relaciones, y una descripción detallada del servidor.

Para mejorar la aplicación y riqueza del universo de AI-LIVE se han añadido nuevas necesidades básicas a los actores entre las que se incluyen:

- "Solitude" : contabiliza la soledad del actor y sirve para regular su necesidad de interactuar con otros actores del escenario o usar algún objeto del mismo para ello en el caso de que sea el único actor conectado al servidor.

- "thirst" : representa la sed del actor y le empuja a beber o comprar objetos de bebida de las tiendas.

- "catchedObjects" : no es una necesidad en sí sino una lista con objetos que carga el actor. Para limitar el peso y volumen que pueden cargar los actores se han añadido nuevos slots a las clases apropiadas.

Se han eliminado algunas características anteriores como la bolsa que se utilizaba para recoger objetos del escenario. Esta bolsa era un objeto de tipo contenedor para almacenar distintos objetos que los actores podían utilizar. La bolsa se asignaba al conectarse un cliente y no existían acciones para coger objetos del escenario ni dejar los que hubiera en la bolsa. Este elemento limitaba el número de actores en un escenario ya que un cliente sin una bolsa asignada con determinados objetos no podía realizar diversas acciones que utilizaran dichos objetos.

También se ha modificado el drive o indicador que mide la felicidad del actor para hacerlo dependiente del estado real del mismo en cada momento. La felicidad de un actor viene dada por la diferencia entre el valor actual de todos sus drives con sus respectivos máximos. En la siguiente tabla se muestra la relación:

drive	boredom	solitude	hunger	thirst	tiredness	dirtness
	31	22	27	54	23	76
	30	30	30	30	30	30
maxdrive	max-boredom	max-solitude	max-hunger	max-thirst	max-tiredness	max-dirtness
drive*100/maxdrive	103	73	90	180	76	253
Satisfaction normalizado (intervalo[0-100] drive*100/maxdrive)	100	73	90	100	76	100
Welfare(100-satisfaction normalizado)	0	27	10	0	24	0
Valor welfare final (sumatorio satisfaction/6)	10					

Tabla 2: Cálculo de la felicidad

Primero se calcula el porcentaje del valor de cada drive respecto al máximo definido en el perfil del actor. Si el drive está por encima del máximo definido para el actor, se considera que el porcentaje es 100%. Como la felicidad es un valor positivo al igual que el resto de drives, se calcula el valor inverso por lo que si todos los drives son 0, se considera que las necesidades del actor están completamente satisfechas y su felicidad es 100%. En caso contrario, se calcula la media de satisfacción de todas sus necesidades y en el caso mencionado de que un drive esté por encima de su valor máximo, se considera que ese drive aporta un 0% de felicidad para calcular el valor final.

En el ejemplo anterior se muestran los valores de los drives para un actor y sus máximos definidos en el perfil, con los cálculos intermedios que se realizan y el valor final obtenido para el drive "welfare" que representa la felicidad del actor.

Para simular el paso del tiempo se ha añadido la posibilidad de incrementar el número de turnos que una acción puede durar de forma que un actor puede ejecutar una acción determinada y pasarse los siguientes turnos con la misma acción. En cada turno que el actor ejecuta una acción se actualizan sus drives simulando el paso del tiempo, añadiendo hambre, sed, cansancio y empeorando su higiene. Además si un actor está realizando una acción que dura más de un turno, se siguen actualizando sus drives según corresponda la acción que está ejecutando.

Se ha añadido un factor de probabilidad para los clientes de IA de forma que no siempre continúan realizando la acción que dura

más de un turno sino que dependiendo de la probabilidad definida en su perfil, unas veces continuará ejecutando la acción que perdura varios turnos y otras ejecutará una nueva acción. Esto se ha hecho para provocar distintos comportamientos en los clientes de IA y equiparar la posibilidad que tienen los clientes manuales de seguir ejecutando la misma acción durante varios turnos o realizar otra distinta.

Para integrar el motor emocional ya desarrollado, se han mantenido las acciones correspondientes a la comunicación entre actores y se ha usado el sistema de gustos, extendiéndolo a todos los objetos del escenario de forma que un actor siempre intentará usar los objetos que más le gustan antes que el resto. De esta forma se consiguen distintos comportamientos dependiendo de los gustos del actor y el escenario dado.

Para portar la aplicación a sistemas *Windows* se ha buscado un compilador compatible y se ha modificado el código fuente donde ha sido necesario para poder compilar bajo ese entorno. El compilador elegido ha sido *MinGW* por ser el más parecido a las herramientas que se emplean en entornos *Unix*. Se ha creado un archivo que permite la compilación en ambas arquitecturas de forma simple.

Se ha actualizado el código fuente de *CLIPS* a la última versión disponible en la página web del proyecto y se ha adaptado el proyecto a los cambios introducidos en esta versión incluyendo los necesarios en el script de compilación y adaptando algunas funciones añadidas al motor de *CLIPS* para AI-LIVE.

El ciclo de vida empleado en el desarrollo ha sido el modelo iterativo incremental con prototipos. Se han identificado los requisitos de cada punto de mejora y utilizando el sistema de control de versiones se subían los cambios con la nueva funcionalidad implementada.

Los prototipos del ciclo de vida se han implementado con cada mejora durante la fase de desarrollo, por ejemplo al introducir nuevos drives o medidores y sus correspondientes acciones asociadas. Se realizaban distintas pruebas con los clientes para comprobar su comportamiento ante la nueva acción disponible y se ajustaban los elementos del escenario para conseguir el resultado deseado que era la realización de forma automática de las acciones cuando los drives o indicadores de los actores alcanzaban los umbrales definidos.

Cada iteración con su prototipo se actualizaba en el sistema de control de versiones quedando así disponible para su prueba e integración con otros clientes.

5.2 Arquitectura de la aplicación

En el proyecto se ha utilizado la arquitectura cliente-servidor en el que los distintos clientes de IA o un cliente manual controlado por un jugador se conectan a un mismo servidor. El servidor recibe las acciones de cada cliente y es el que contiene la información del mundo virtual y su estado, de forma que solamente el servidor es el que puede modificar realmente el universo virtual simulado, el resto de clientes recibe la actualización con los cambios que envían otros clientes al servidor.

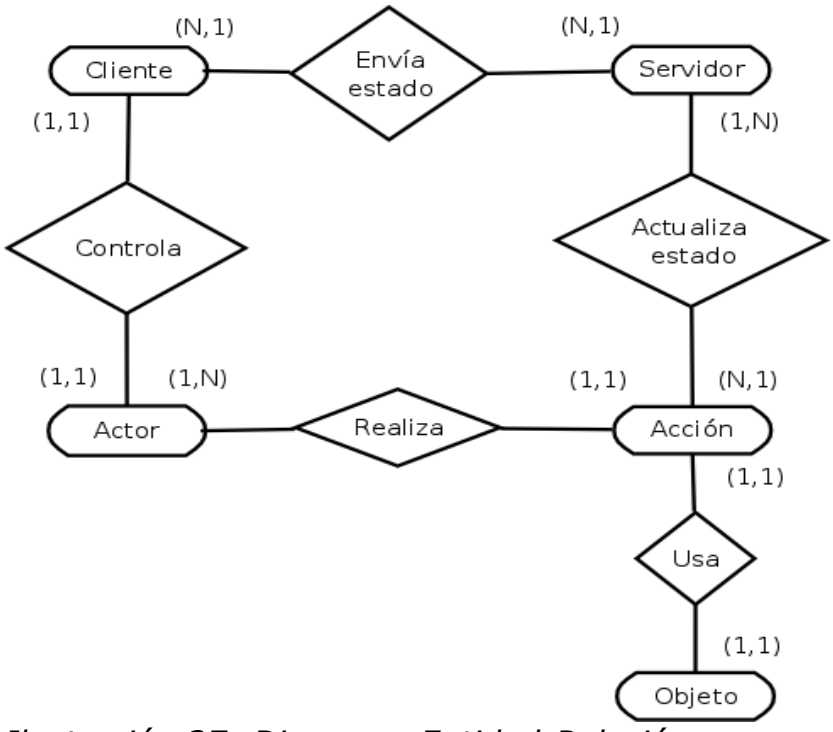


Ilustración 27: Diagrama Entidad-Relación

La aplicación AI-LIVE utiliza una comunicación entre los módulos mediante *sockets*, sobre el protocolo TCP/IP. Se puede realizar tanto desde un mismo computador o desde distintos computadores. La elección de esta arquitectura permite repartir la capacidad de proceso entre los clientes y los servidores, ya que facilita la separación de responsabilidades y la centralización de la gestión de la información.

De tal manera, el servidor tiene toda la información del juego, que transmite a cada cliente, y de esta forma se reparte la capacidad de proceso entre los módulos.

Los tipos de clientes de la aplicación son varios (que serán explicados más adelante): los clientes de IA, divididos en los clientes *CLIPS* (sistema basado en reglas) y PRODIGY (un planificador de tareas), cliente que contiene la interfaz gráfica (GUI), y cliente manual, que permite al usuario interaccionar con el juego eligiendo él mismo las acciones que desea.

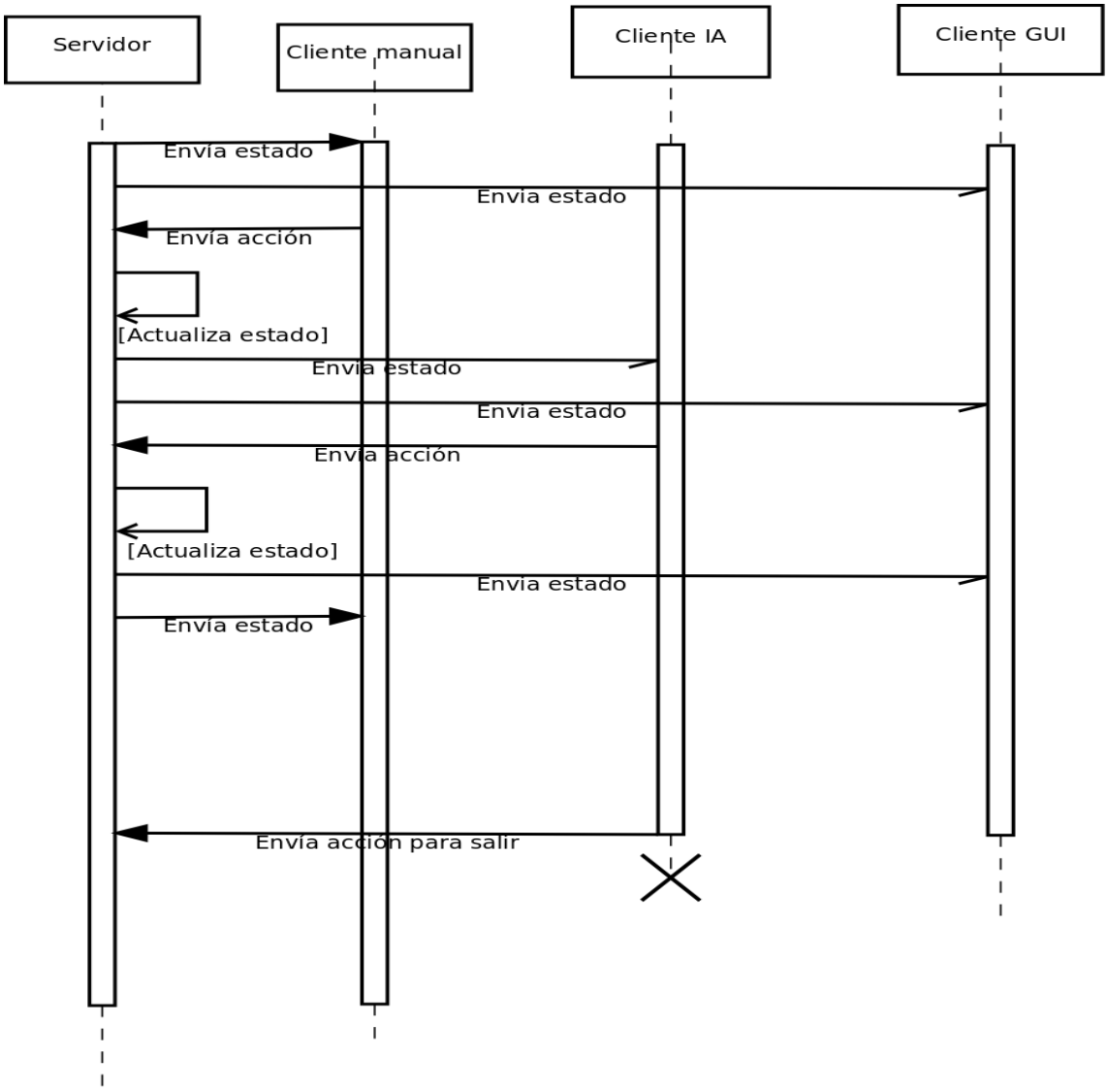


Ilustración 28: Diagrama de secuencia general

En la ilustración 19 se muestra un diagrama de secuencia general donde se observa la asignación de turnos por parte del servidor a cada cliente, recibiendo después la acción de cada uno de

ellos y actualizando el escenario tras la ejecución de estas acciones. El cliente gráfico GUI recibe en cada turno el estado actualizado para representar cada cambio que se realiza en el escenario.

En AI-LIVE, cada módulo de cliente, exceptuando el cliente GUI, representa a un actor o personaje que habita en un entorno y que realiza una serie de acciones disponibles. Una vez realizada la acción manda su petición al servidor, que es quien en realidad hace efectiva la acción del cliente, y actualiza el estado del universo (cambios del escenario, atributos y características del cliente, objetos, etc). Posteriormente el servidor envía el nuevo estado al resto de clientes para que decidan su siguiente acción. La diferencia radica en que los clientes de IA toman sus decisiones solos, mientras que el cliente manual indica la acción que desea realizar a través de la opción que ha elegido el usuario.

Los distintos módulos son independientes entre sí y cada uno utiliza su propio motor de IA, incluido el cliente manual pero los cambios que realizan en el estado del mundo virtual no tienen influencia en el resto de clientes, solamente el servidor mantiene el estado válido y se envía a cada cliente al comienzo de su turno, sincronizando así el fichero de estado entre todos los clientes.

Con la siguiente figura observaremos el conjunto de módulos que forman la arquitectura utilizada, sirviéndonos de introducción para el siguiente apartado del capítulo.

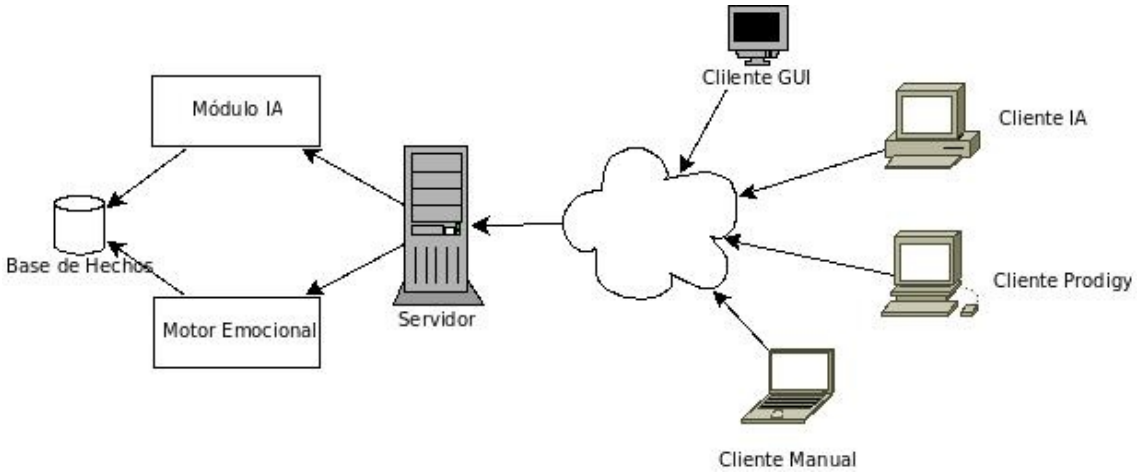


Ilustración 29: Arquitectura AI-LIVE cliente-servidor

5.3 Módulos de la aplicación

5.3.1 Servidor

El servidor una vez iniciado espera continuamente peticiones de conexión de los clientes. Una vez llegada una petición, el cliente pasa a formar parte de la lista de clientes para asignarles un turno, mediante el algoritmo de planificación Round Robin. El servidor contiene el estado completo del juego, que es actualizado al ejecutar la acción pedida por el cliente y al finalizar su turno, transmitiendo el estado actual al siguiente cliente. Los módulos que componen el servidor son:

- **Módulo principal**: se encarga de establecer las conexiones con los clientes, llevar a cabo el protocolo de comunicaciones, asignar los turnos de éstos, y hacer llamadas al módulo de inteligencia artificial.

- **Módulo de IA**: contiene el estado completo del juego, y ejecuta las acciones recibidas por los clientes, actualizando el estado.

- **Módulo del motor emocional**: controla las emociones, gustos y relaciones de los actores cuando tiene lugar una acción comunicativa.

5.3.2 Clientes

Cuando un cliente se conecta al servidor, se le asigna un socket con el que pueden intercambiarse información siguiendo el protocolo de comunicación establecido. Los módulos que forman los distintos clientes son:

- Módulo principal: es el encargado de establecer la conexión con el servidor y de llevar realizar las comunicaciones a través del protocolo de comunicaciones.

- Módulo de ejecución: realiza la funcionalidad específica de cada cliente, dependiendo de su tipo. Actualmente existen cuatro tipos de clientes:

  - **Cliente *CLIPS***: es una entidad capaz de elegir la acción que quiere realizar e interactuar con otros clientes y objetos. Está construido con un sistema basado en reglas.

- **Cliente Prodigy**: al igual que el cliente *CLIPS*, utiliza un motor de IA, pero éste integra un planificador de tareas llamado *Prodigy*, en lugar de emplear *CLIPS*.

- **Cliente manual**: se basa en el cliente *CLIPS*, pero la diferencia es que la toma de decisión de un acción la realiza una persona física, que es la que interactúa con el juego e indica al actor la acción.

- **Cliente GUI**: es el encargado de representar gráficamente el estado del juego. A diferencia de los clientes basados en IA y el manual, éste no selecciona ni envía la acción a realizar por el servidor.

Una vez el servidor ha enviado el estado al cliente al que le corresponda el turno, ya sea de IA o manual, tendrá un tiempo ilimitado para decidir la acción que desea ejecutar, para después enviársela al servidor. Mientras, los otros clientes permanecerán a la espera de su turno.

Los clientes GUI, en cambio, no tienen que esperar, ya que deben actualizar, en los turnos de los demás clientes, la información gráfica recibida en el estado del juego que le envía el servidor.

5.4 Protocolo de comunicaciones entre los módulos

Para la comunicación entre los módulos de la aplicación, existe un protocolo que utiliza una serie de etiquetas que identifican el contenido o la función de cada mensaje que intercambian el servidor y los clientes:

– **HOLA**: es utilizada por todos los módulos de la aplicación. Sirve para iniciar la conexión entre ellos y en él se indica la versión y las diversas opciones soportadas por los elementos que intervienen en la comunicación.

– **STAG**: utilizada por los clientes para indicar el escenario en donde empezarán su ejecución.

– **ACTR**: los clientes, tanto de IA como los manuales, utilizan esta etiqueta para especificar el identificador del actor que están controlando.

– **ACTN**: es utilizada por los clientes para indicar la acción que desean realizar, y que ejecute el servidor.

– **STAT**: el servidor la utiliza para enviar el estado del juego a un cliente.

– **GO**: es utilizada por el servidor y el cliente GUI para indicar que el módulo actual ha terminado la actualización de la información y que el otro módulo puede continuar su ejecución.

– **Ex**: se utiliza con el cliente GUI para indicar al servidor que ha finalizado su ejecución.

Se han reescrito algunas de las funciones que implementaban el protocolo de comunicaciones mediante el uso de sockets para asegurar la compatibilidad con otros sistemas y se ha adaptado el código para poder compilarlo en un entorno *Windows*.

5.5 Entradas y salidas de los módulos

En este apartado indicaremos las entradas necesarias que recibe cada módulo para su inicialización y posterior ejecución, así como las salidas que produce.

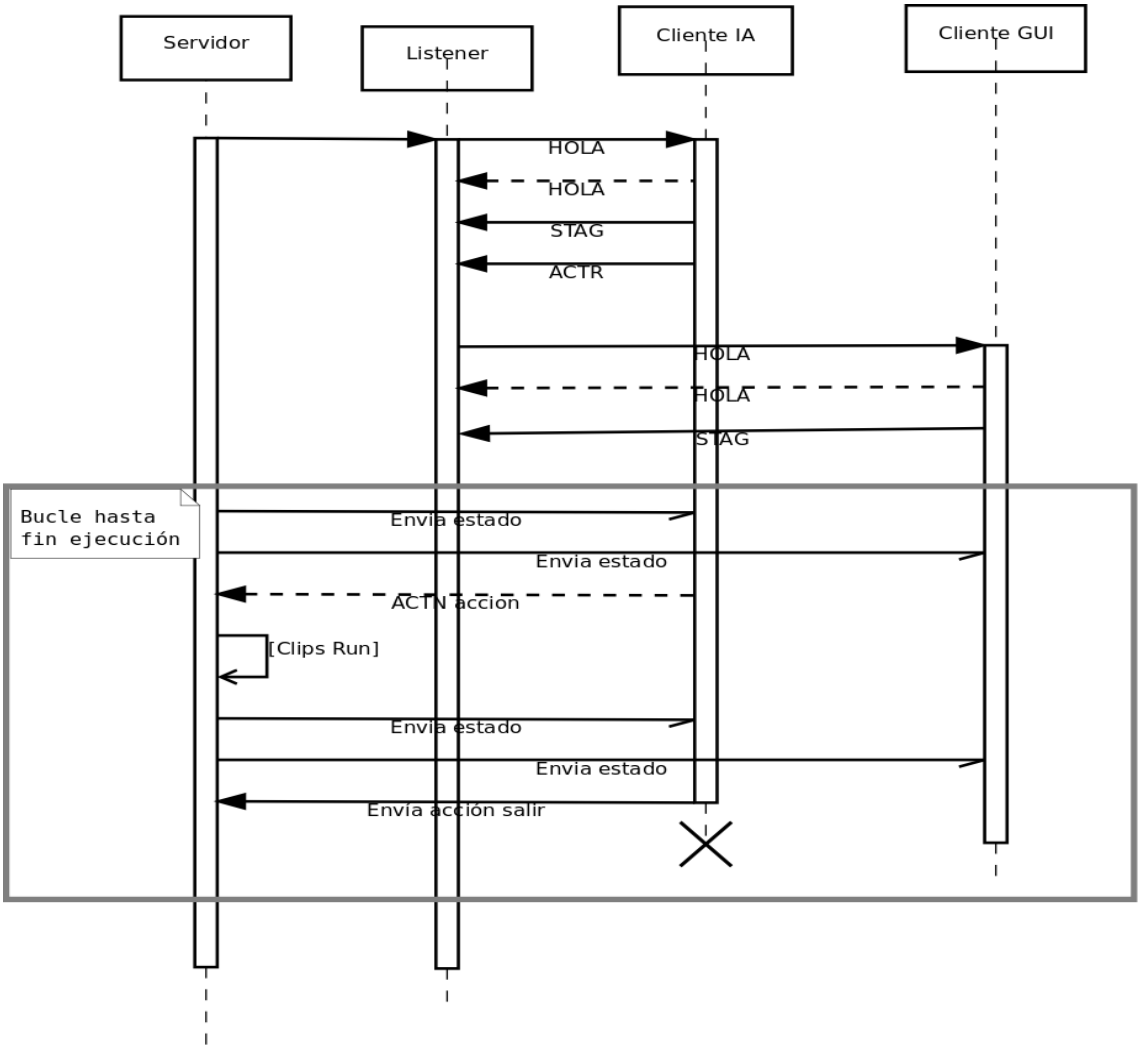
5.5.1 Servidor

5.5.1.1 Entradas

- Ontología (ontology.clp)
- El código del servidor *CLIPS* (server.clp)
- Código del motor emocional (emotional_engine_server.clp)
- Estado inicial del juego (initial.state)

5.5.1.2 Ejecución normal

Ilustración 30: Diagrama de secuencia ejecución normal



El diagrama de la ilustración 21 muestra el protocolo de comunicaciones entre el servidor y los distintos módulos.

Se muestra el envío de acciones por parte de los clientes y el envío del fichero de estado que realiza el servidor tras ejecutar las acciones que recibe. Se muestra que un cliente puede desconectarse en cualquier momento y el servidor puede seguir atendiendo peticiones de conexiones de otros clientes de forma paralela.

El servidor contiene todo el estado del universo, dividido en los diferentes escenarios existentes. Se encarga de seleccionar el conjunto de información que recibirá cada cliente, guardarlo como un fichero cuyo nombre se corresponde con el indicador asignado al realizar la conexión (IdCliente.state), y enviarlo a su destinatario

como un mensaje con la etiqueta STAT. Esta información contiene el estado de los objetos y su posición en el escenario actual, y los atributos del actor que controla el cliente.

Para el cliente GUI el servidor almacena igualmente la información en un fichero, pero que contiene la información del escenario y su nombre se corresponde con el identificador del escenario actual en el que se encuentra el actor en ese turno.

Posteriormente envía el mensaje con la etiqueta GO para que el cliente GUI continúe su ejecución, y a continuación le envía el estado a través de un mensaje con la etiqueta STAT como se ha mencionado antes. La información del mensaje contiene el estado de los objetos del escenario, los actores que se encuentran en ese escenario en ese momento, y la acción que va a realizar el actor actual.

Para un cliente IA el servidor envía sólo la información sobre el escenario donde se encuentra, junto con los valores de los atributos del actor. Pero no recibirá el estado de otros agentes. También, si existen otros escenarios en el juego a los que puede acceder, el servidor le enviará también una información reducida de los objetos que puede encontrar allí, para saber que la acción que quiere realizar la puede hacer en otro escenario.

El servidor recibirá de los clientes, salvo del cliente GUI, mensajes ACTN con la acción que quiere realizar el actor que controla el cliente.

Cuando la acción que recibe el servidor es de tipo comunicativa (un actor desea hablar con otro), el módulo de IA del servidor realiza una llamada al motor emocional, para actualizar las emociones de los actores, sus relaciones y gustos.

5.5.2 Clientes CLIPS y manual

5.5.2.1 Entradas

- Ontología (ontology.clp)
- Código del cliente *CLIPS* (client.clp)
- Perfil del actor (DEFAULT_ACTOR.profile)

5.5.2.2 Ejecución normal

El cliente genera un archivo de traza, (id_actor-TRAZA.txt, donde id_actor es el identificador asignado al realizar la conexión con el servidor), para ir almacenando las acciones que va realizando.

Antes de realizar una acción, recibe del servidor a través de un mensaje con la etiqueta STAT, el estado actual para que el actor decida la acción a realizar.

Posteriormente selecciona la acción (decidida por el motor de IA si es cliente de IA, o manualmente si es el cliente manual) que quiere ejecutar, y se almacena en un fichero (current.action), que envía al servidor en un mensaje con la etiqueta ACTN.

5.5.3 Cliente Prodigy

5.5.3.1 Entradas

- Dominio del problema (domain.lisp)
- Problema a resolver (problem.lisp)
- Perfil del actor (DEFAULT_ACTOR.profile)

5.5.3.2 Ejecución normal

El cliente almacena la acción a realizar en un archivo (current.action), el cual enviará al servidor a través de un mensaje con la etiqueta ACTN.

Después traduce el estado del juego recibido del servidor de *CLIPS* a *LISP* para poder manejar el problema. A continuación genera el objetivo del problema, que es obtener el mayor número de objetos que hay en el escenario. Al igual que los clientes *CLIPS* y manual, éste recibe del servidor la información relevante del estado actual del juego, a través de un mensaje con la etiqueta STAT.

5.5.4 Cliente GUI

5.5.4.1 Entradas

- Núcleo de OGRE
- Plugins (CEGUI y OIS)
- Recursos de imagen necesarios (objetos, texturas, animaciones...)

5.5.4.2 Ejecución normal

El cliente recibe del servidor un mensaje con la etiqueta STAT en el cual contiene la información del estado actual del juego, y lo almacena en un archivo (gui.state). Después de recibir el mensaje, accede al archivo para leerlo y crea, modifica o elimina las entidades gráficas del escenario actual, según el contenido del estado recibido.

Una vez generado el escenario gráficamente, el cliente GUI enviará al servidor la etiqueta GO para indicarle que ha terminado de actualizar la información gráfica, y que éste puede continuar con su ejecución.

Cuando el cliente GUI ha terminado su ejecución, enviará la etiqueta EX al servidor para indicarle que le elimine de la lista de clientes, para no enviarle los posteriores estados actualizados.

5.6 Modelo de conocimiento de la aplicación

A continuación se muestra un gráfico con el modelo de conocimiento de la aplicación. Toda la ontología con las clases y objetos usada en *CLIPS*.

Las clases más importantes son:

- **ClientAction**: Clase para definir todas las acciones posibles que pueden ejecutar los actores en los distintos escenarios y los objetos que intervienen en ellas. De ella heredan todas las acciones posibles que los actores pueden realizar:

- **PutDownAction**: con esta acción el actor deja en el escenario un objeto que tenga en su poder.

- **PickupAction**: con esta acción el actor coge un objeto del escenario que podrá utilizar después.

- **Continue**: con esta acción, el actor comunica al servidor que decide continuar con la acción del último turno, siempre que haya hecho una acción que puede durar más de un turno.

- **MoveAction**: el actor se desplaza de una celda a otra vacía del escenario.

- **Read**: el actor lee un libro o revista que tenga en su poder.

- **Resting**: utilizar un objeto de descanso para mejorar el drive de cansancio, disminuyéndolo.

- **Washed**: utilizar un objeto de higiene personal para disminuir el drive suciedad.

- **Play**: utilizar un objeto de ocio que posea el actor.

- **PickupFood**: Coger y alimentarse con algún alimento del frigorífico.

- **PickupDrink**: beber algo del frigorífico para saciar la sed del actor.

- **BuyFood**: comprar comida en la tienda y alimentarse con ella.

- **BuyDrink**: comprar bebida en la tienda y saciar la sed del actor.

- **Exit**: salir del escenario. Esta acción no se envía desde el motor de *CLIPS* sino que es enviada automáticamente al cerrar el cliente con "ctrl+c" o con la opción correspondiente en el cliente manual.

- **Comunication**: acción que activan los actores cuando sienten la necesidad de comunicarse con otro actor y no hay ninguno disponible en el escenario, utilizando pues un objeto de tipo ComunicationObject.

- **Working**: Usar un objeto de trabajo del escenario para conseguir riqueza y poder comprar objetos con ella.

- **FindFreeCell**: desplazarse a una celda libre del escenario.

- **ChangeStage**: cambiar de escenario.

- **AddClient**: esta regla se envía al conectar el cliente al servidor, indicando los atributos definidos en su perfil.

- **Entity**: clase de la que heredan todos los objetos, tanto los del escenario con los que interactúan los actores y realizan acciones con ellos como otros auxiliares como son los decorativos o muros del escenario que lo limitan y con los que el actor no puede interactuar.

- **Object**: objetos del escenario que el actor puede recoger y usar. De esta clase heredan todas las subclases de objetos y de estas, los objetos en sí que utilizan los actores y se definen en escenario:

- **Container**: contenedor de objetos, de la que heredan el frigorífico y la tienda.

- **ComunicationObject**: son objetos que el actor utiliza para comunicarse con otros actores si no puede hacerlo directamente, por ejemplo cuando solo hay un actor conectado a un escenario.

- **MiscellaneousObject**: son todos los objetos decorativos del escenario que no están diseñados para que el actor interactúe con ellos.

- **ReadingObject**: diversos objetos de lectura.

- **WorkObject**: objetos de trabajo en el escenario con los que el actor puede obtener riqueza.

- **BathObject**: objetos para mejorar la higiene de los actores.

- **LeisureObject**: objetos de ocio para entretener al actor.

- **RestObject**: distintos objetos que sirven para que un actor descanse.

- **Teleporter**: objetos que se utilizan para cambiar de escenario, simulando la funcionalidad de una puerta.

- **Food**: comida para alimentar a los actores.

- **Drink**: bebida para saciar la sed de los actores.

- **Actor**: Los actores del universo virtual, tanto controlados de forma autónoma, como por un jugador humano o controlados por otro sistema de inteligencia artificial especialmente preparado para interactuar con los datos e instancias de *CLIPS*.

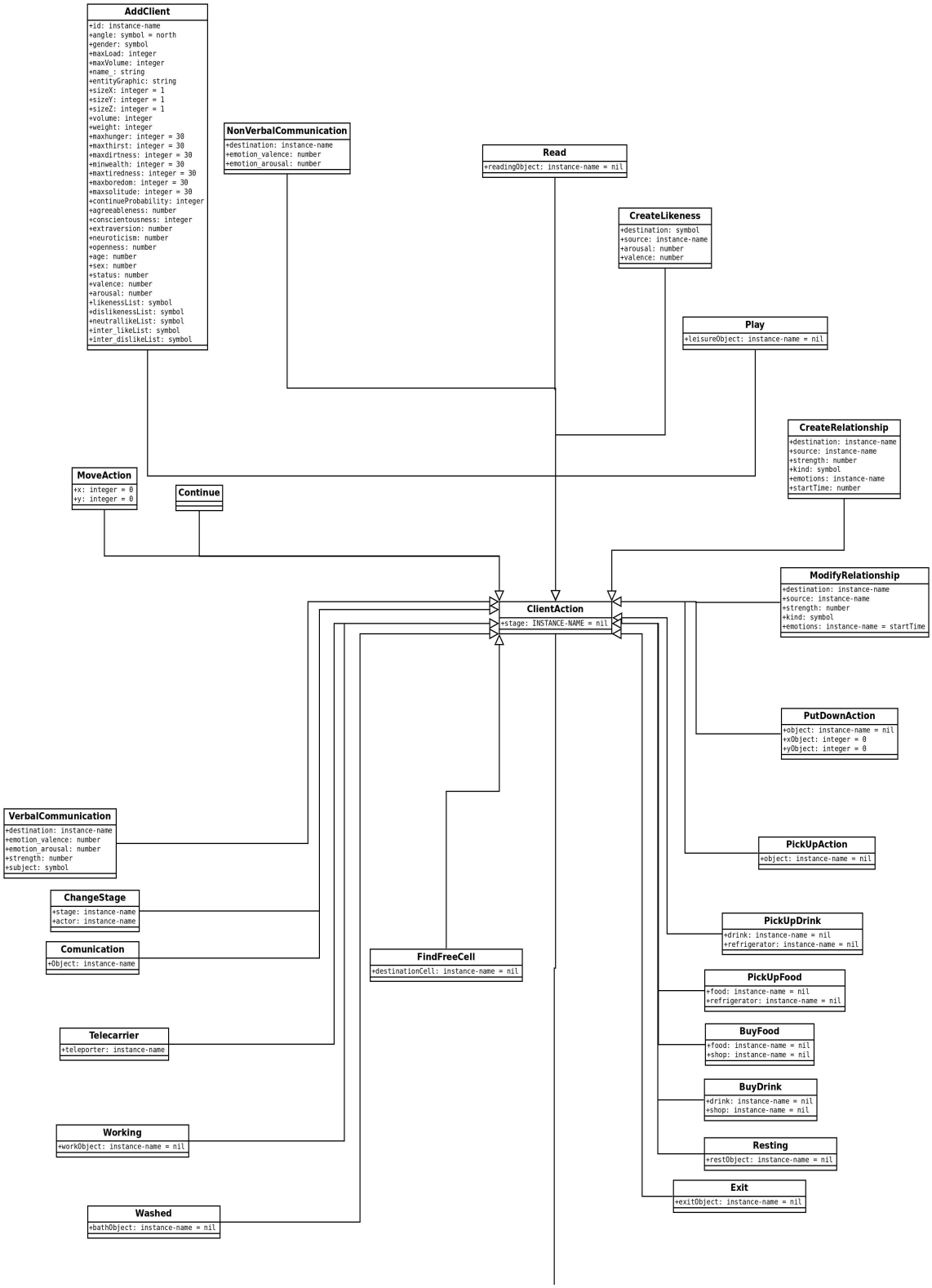


Ilustración 31: Modelo de conocimiento, parte 1

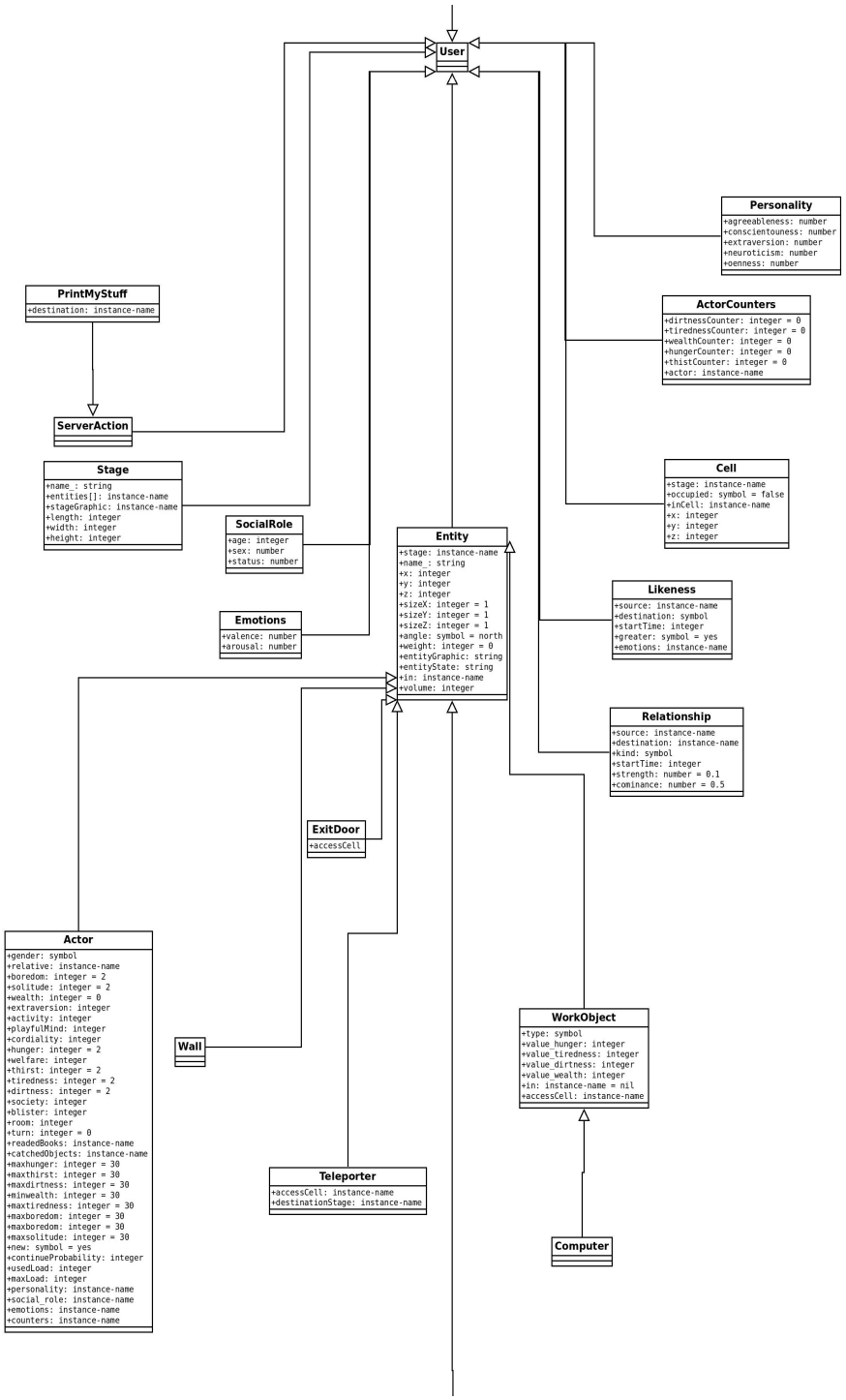


Ilustración 32: Modelo de conocimiento, parte 2

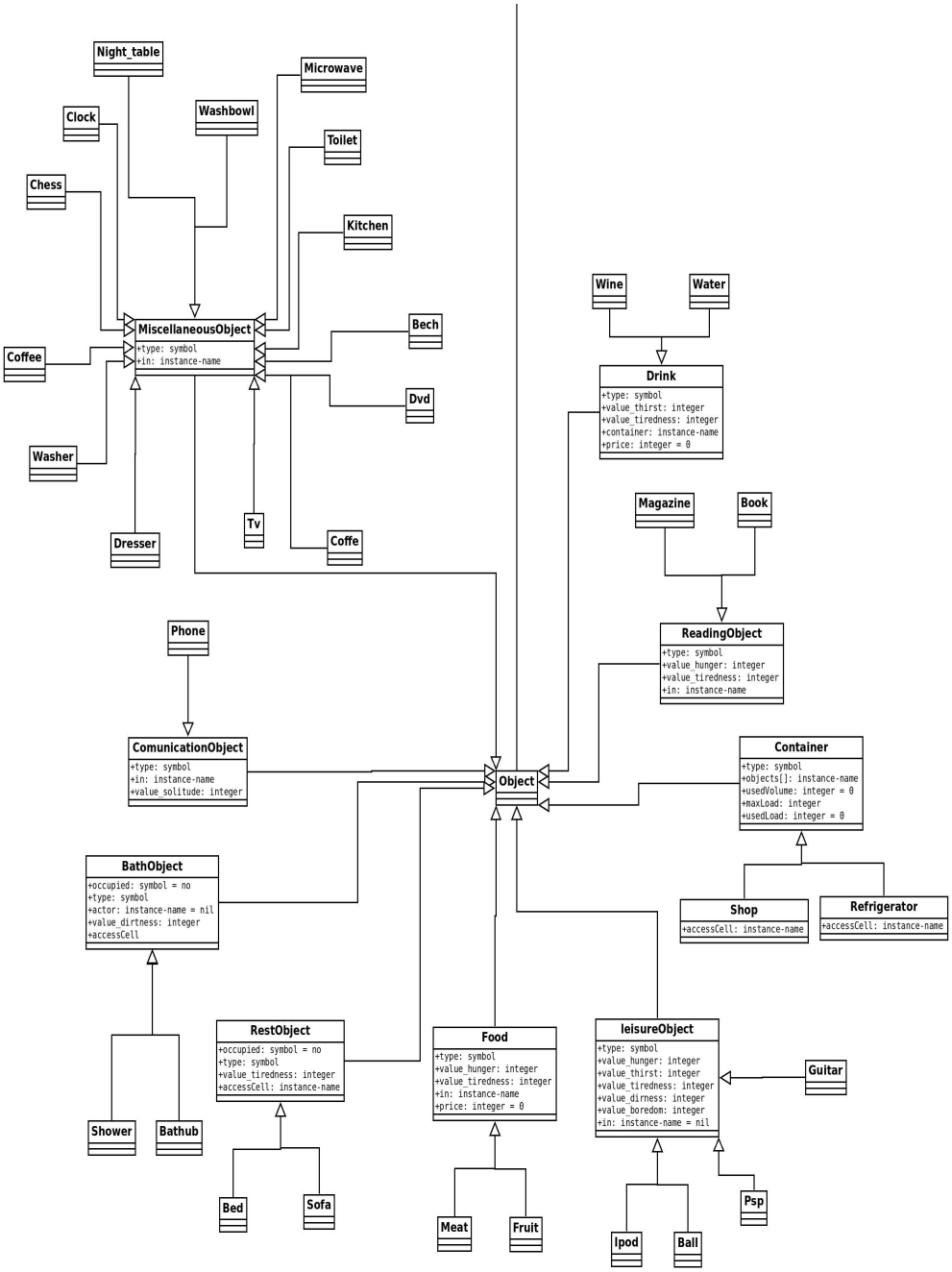


Ilustración 33: Modelo de conocimiento, parte 3

5.7 Descripción detallada del servidor

En este apartado se describe de forma detallada el funcionamiento del servidor. Los módulos del servidor son el módulo principal, el motor de inteligencia artificial y el motor emocional.

5.7.1 Módulo principal

Este módulo se encarga de gestionar las conexiones con los distintos clientes, la asignación de turnos y las llamadas al módulo de inteligencia artificial. Después de la ejecución de la acción de un actor, se genera el estado y se envía a los clientes correspondientes.

Para gestionar las conexiones con los clientes, se crea un hilo de ejecución en el que se espera la conexión de un cliente y la posterior negociación. Una vez que se acepta al cliente, se añade a la lista de clientes para concederle el turno y recibir las acciones que envíe.
Este módulo carga los archivos esenciales para el módulo de inteligencia artificial que son:

- **Ontología**: contiene el conocimiento de todas las clases y sus relaciones en el universo virtual de AI-LIVE, desde los objetos más básicos hasta las acciones más complejas posibles de los actores.

- **Estado inicial**: es un conjunto de instancias que definen un escenario de partida donde se especifican todos los elementos del escenario que lo definen, incluyendo las distintas estancias, paredes que las delimitan y objetos con los que pueden interactuar los actores.

- **Código fuente de la IA del servidor**: contiene las reglas, métodos e instancias que se necesitan para que funcione el motor de inteligencia artificial.

El servidor puede comunicarse con distintos tipos de clientes y dependiendo del tipo de cliente conectado, varía el comportamiento del servidor:

- **Cliente de inteligencia artificial**: el servidor le asigna el turno, le envía el estado actual y espera recibir una acción del mismo. A continuación se ejecuta la acción y se actualiza el estado del universo virtual. Después de la finalización del turno, el cliente pasa a la espera hasta que el servidor le conceda de nuevo el turno y le envíe el estado actualizado.

- **Cliente manual**: es un caso especial de cliente de inteligencia artificial en el que un humano elige qué acción ejecutar en cada momento. La asignación de turno y posterior espera es igual que para un cliente de IA.

- **Cliente GUI**: este tipo de cliente no tiene que esperar a que el servidor le conceda el turno, siempre recibe el estado actualizado del universo virtual después de la acción de cada actor. Este cliente tampoco envía ninguna acción al servidor pero hay una espera por parte del servidor para que de tiempo a visualizar todos los cambios en la pantalla entre cada acción de los actores.

5.7.2 Motor de inteligencia artificial

Este módulo se encarga de la inteligencia artificial en el lado del servidor donde se procesan las acciones que envían los clientes y se actualiza el estado del universo virtual.

En el servidor existe una regla equivalente a las existentes en los clientes para asegurar la posible ejecución de todas las acciones. Además contiene reglas propias para mantener el estado del universo virtual actualizado y para realizar determinadas acciones como la actualización de los drives de los actores y gestionar la conexión de los actores, creando todas las instancias relacionadas con el actor y su desconexión del mundo virtual, eliminando las instancias correspondientes.

El motor de inteligencia artifical embebido en el servidor carga además el fichero de reglas correspondiente al módulo del motor emocional y sus acciones asociadas para expresar sentimientos y la comunicación entre actores.

Para ver una descripción detallada de las reglas y métodos implementados, consultar el punto 12.1 Manual de referencia.

Se ha introducido el concepto de tiempo añadiendo a las acciones de los actores la posibilidad de una duración mayor a un turno. En las acciones como descansar o trabajar el servidor modifica la instancia del actor para indicarle que puede seguir realizando la misma acción en turnos sucesivos o realizar otra distinta. El cliente tiene una acción llamada "continue" para indicar al servidor que decide continuar con la última actividad iniciada. El servidor se encarga de actualizar consecuentemente los drives del actor. Por ejemplo si un actor decide descansar en un sofá, enviará la acción de descansar al servidor utilizando el objeto sofá para reducir su cansancio y en los siguientes turnos tiene la opción de enviar la acción de continuar, sin especificar el objeto usado y el servidor seguirá actualizando los drives del actor utilizando los valores del objeto sofá de la primera acción.

También se simula el paso del tiempo en el servidor añadiendo un desgaste periódico en los actores en cada turno, de esta forma se les obliga a realizar diversas acciones para mejorar la situación de sus drives.

Se han implementado las acciones de coger y dejar objetos del escenario ya que anteriormente existía un objeto llamado bolsa que era un contenedor de objetos y los actores usaban los objetos de este contenedor pero no podían coger nuevos objetos para dejarlos en esa bolsa ni dejar objetos de la propia bolsa en el escenario. Este sistema limitaba el número de actores de un escenario, ligando su número a la cantidad de bolsas de objetos que hubiera definidas en el escenario ya que diversas acciones de los actores requerían el uso de objetos de esas bolsas y si un actor no tenía ninguna asignada, no podía realizar la acción en concreto.

Se ha eliminado el uso de la bolsa y se ha implementado una lista de objetos que mantiene en su poder un actor, limitando los objetos que puede cargar por el peso total de los mismos. Algunas acciones que requieren el uso de ciertos objetos solo pueden realizarse si el actor tiene en su poder ese objeto por lo que antes debe realizar la acción de desplazarse y coger el objeto en cuestión. Otras acciones como alimentarse, beber o hacer uso de algún otro objeto del escenario no requieren que el actor posea el objeto en un momento determinado sino que utilizan una celda de acceso para asegurar el uso exclusivo del objeto.

5.7.2.1 Motor emocional

Este módulo se compone de una serie de funciones y métodos escritos en lenguaje C para añadir al motor *CLIPS* varias funciones

que intervienen en la gestión de las emociones de los actores y un fichero de reglas con las acciones definidas en lenguaje *CLIPS*, con las acciones que pueden realizar los actores para comunicarse entre sí. Para una descripción detallada de las acciones y funciones disponibles, consultar el manual de referencia.

Se ha integrado el motor emocional introduciendo un nuevo drive a los actores llamado "solitude". Este drive controla la necesidad que tiene un actor de comunicarse con otro y crece con el tiempo. Esta aproximación presentaba un problema si únicamente había un actor conectado en el escenario, haciéndole imposible comunicarse con otro para mejorar el drive "solitude". Para solucionarlo, se han introducido acciones y objetos que simulan un teléfono con el que los actores pueden mejorar este drive en el caso de no haber disponible otro actor en el escenario con el que comunicarse directamente.

5.7.3 Ejecución de los módulos del servidor

A continuación se detallan los pasos que se realizan en la ejecución del servidor:

1. Inicialización del servidor: se cargan los ficheros esenciales para la ejecución, se generan los ficheros de estado de los escenarios definidos en el estado inicial y se inicializa el motor de *CLIPS*.

2. Creación del "listener": se crea un hilo para atender las peticiones de conexión de los clientes con la configuración indicada por parámetros. Esta configuración se trata de:

    - **Ruta a la ontología**: fichero que contiene la ontología como se ha descrito anteriormente.

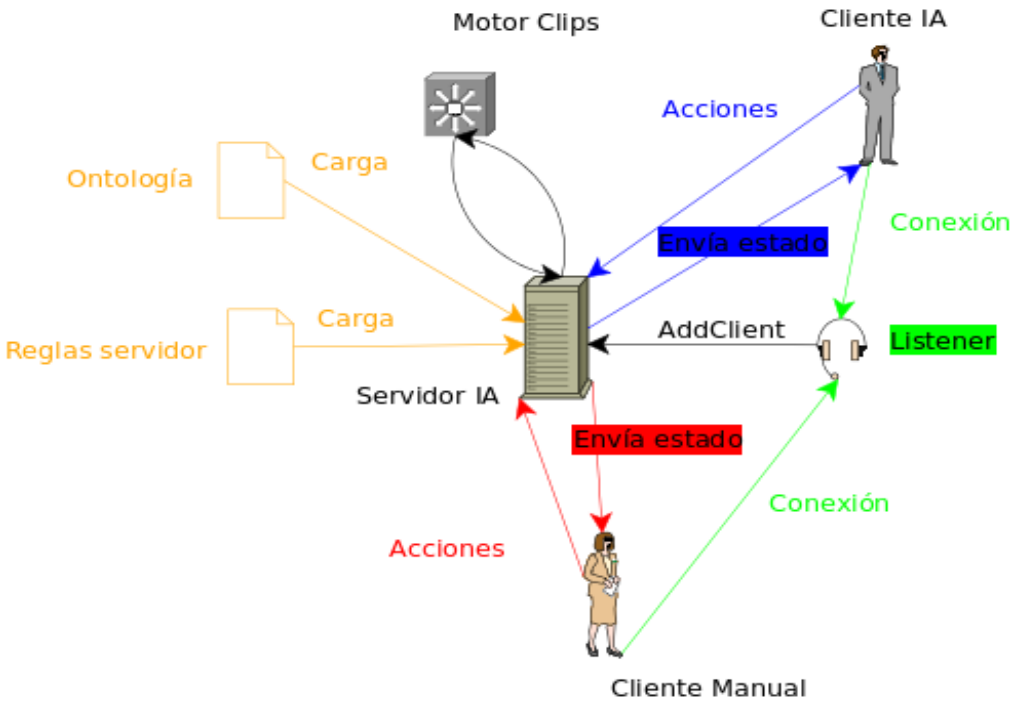
    - **Estado inicial**: contiene la definición de los escenarios del universo virtual.

    - **Puerto**: el puerto de escucha en el que el servidor atiende las peticiones de conexión.

Cuando se produce una conexión de un cliente, se implementa el protocolo de comunicación definido anteriormente para aceptar o rechazar al cliente.

3. Se inicia el bucle de ejecución que no terminará hasta la finalización por parte del usuario del servidor. Este bucle consta de los siguientes pasos:

   - Conceder turno a un cliente.
   - Enviar estado actual del universo virtual.
   - Esperar a la recepción de la acción del cliente.
   - Ejecución de la acción recibida.
   - Actualización del estado y concesión del turno al siguiente cliente.



Anteriormente había un script para la ejecución de cada módulo escrito en *PHP* y cualquier cambio en la configuración del servidor como un cambio de puerto obligaba a cambiar todos los scripts del

resto de módulos. Cada script utilizaba rutas relativas para localizar los perfiles de los actores, que son archivos de configuración en sintaxis de *CLIPS* que definen los parámetros de un actor; y el el fichero de la ontología por lo que había que ejecutarlos desde el directorio de cada módulo. Además no se podía cambiar la ruta o directorio donde se guardan los perfiles porque era relativa a la ubicación del ejecutable del módulo. Este diseño obligaba a mantener varios perfiles de actores distribuidos en varios directorios del proyecto AI-LIVE.

Para facilitar la ejecución de los distintos módulos, se ha desarrollado un script escrito en *BASH* que automatiza todo el proceso. Mediante una serie de argumentos se pueden lanzar distintos módulos a la vez como el servidor, cliente IA, cliente manual o el cliente gráfico tridimensional. El script es configurable, permitiendo seleccionar el puerto de comunicaciones que empleará el servidor para atender las peticiones de conexiones, diversas rutas y nombres de ficheros como el de la ontología y algunas opciones para configurar el tamaño de las ventanas.

El script de ejecución permite seleccionar el perfil del actor que se utilizará al iniciar el módulo de un cliente, bien de inteligencia artificial o el manual controlado por un ser humano. Los perfiles han de estar situados en el directorio especificado en el archivo de configuración del script.

También se ha desarrollado un script *Makefile* para facilitar la compilación del proyecto, eliminado la recursividad que existía anteriormente y proporcionando la posibilidad de compilar cada módulo por separado.

6 EVALUACIÓN DE LA APLICACIÓN

En este punto se describen las pruebas que se realizan sobre la aplicación, la definición de las pruebas y el resultado de las mismas.

6.1 Definición de las pruebas

Para medir los resultados del trabajo realizado, se va a realizar un seguimiento de dos actores controlados por la IA en un escenario de AI-LIVE, comprobando la evolución de los drives de cada agente.

La prueba consiste en ejecutar dos actores con distinta configuración inicial de gustos y preferencias, además de unos niveles máximos distintos en sus perfiles para esos drives.

A continuación se muestran los archivos de configuración de los perfiles de dos actores con los que se harán algunas pruebas de ejecución:

Actor1: Amy

```
([DEFAULT_ACTOR] of AddClient
(name_ "Amy")
(gender F)
(entityGraphic "casual woman")
(maxLoad 5)
(maxVolume 10)
(volume 1)
(weight 55)
(agreeableness 1.0)
(conscientousness 1.0)
(extraversion 1.0)
(neuroticism 0.0)
(openness 1.0)
(age 0)
(status 0)
(sex 0)
(arousal 0.0)
(valence 0.0)
(dislikenessList  Meat Bathtub [Working])
(likenessList Cocina [Read])
(neutrallikeList Ball)
(inter_likeList Fruit)
(inter_dislikeList Sofa [Washed])
(maxhunger 40)
(maxthirst 30)
(maxdirtness 30)
(minwealth 100)
(maxtiredness 40)
(maxboredom 40)
(maxsolitude 20)
(continueProbability 25)
```

```
(hcocina 1)
(hmecanica 2)
(hlogica 3)
(hfisica 4)
(teorico 4)
(practico 2)
(program Cocina)
```

Actor2: Mike

```
([DEFAULT_ACTOR] of AddClient
(name_ "Mike")
(gender M)
(entityGraphic "casual man")
(maxLoad 5)
(maxVolume 10)
(volume 1)
(weight 55)
(agreeableness 1.0)
(conscientousness 1.0)
(extraversion 1.0)
(neuroticism 0.0)
(openness 1.0)
(age 0)
(status 0)
(sex 0)
(arousal 0.0)
(valence 0.0)
(dislikenessList [Play] Fruit [Working])
(likenessList Bathtub Sofa [Resting])
(neutrallikeList Book)
(inter_likeList Meat [Washed] Wine)
(inter_dislikeList Bed Shower)
(maxhunger 50)
(maxthirst 40)
(maxdirtness 50)
(minwealth 90)
(maxtiredness 60)
(maxboredom 35)
(maxsolitude 30)
(continueProbability 80)
(hcocina 1)
(hmecanica 2)
(hlogica 3)
(hfisica 4)
(teorico 4)
(practico 2)
(program Cocina)
```

Para comprobar la evolución de los drives, se crea un método en el código *CLIPS* de los clientes para que en cada ejecución de una regla del cliente, guarde en un fichero la situación de todos los drives del actor. El método se llama "printActorCounters" y guarda en un fichero con el ID del actor correspondiente el valor de los drives "hunger", "thirst", "wealth", "dirtness", "tiredness", "boredom",

"solitude", "welfare", "valence" y "arousal". También se guarda la acción que ejecuta en cada turno el actor.

6.2 Realización de las pruebas

Para medir los resultados del trabajo realizado, se va a realizar un seguimiento de dos actores controlados. Se ejecutan los dos actores con los perfiles configurados como se ha especificado en el apartado anterior y se guardan los valores de sus drives en un fichero.

Con los valores guardados, se generan las siguientes gráficas de evolución:

Mike, 85 ejecuciones, valores tomados en las acciones 1,5,10,15....85

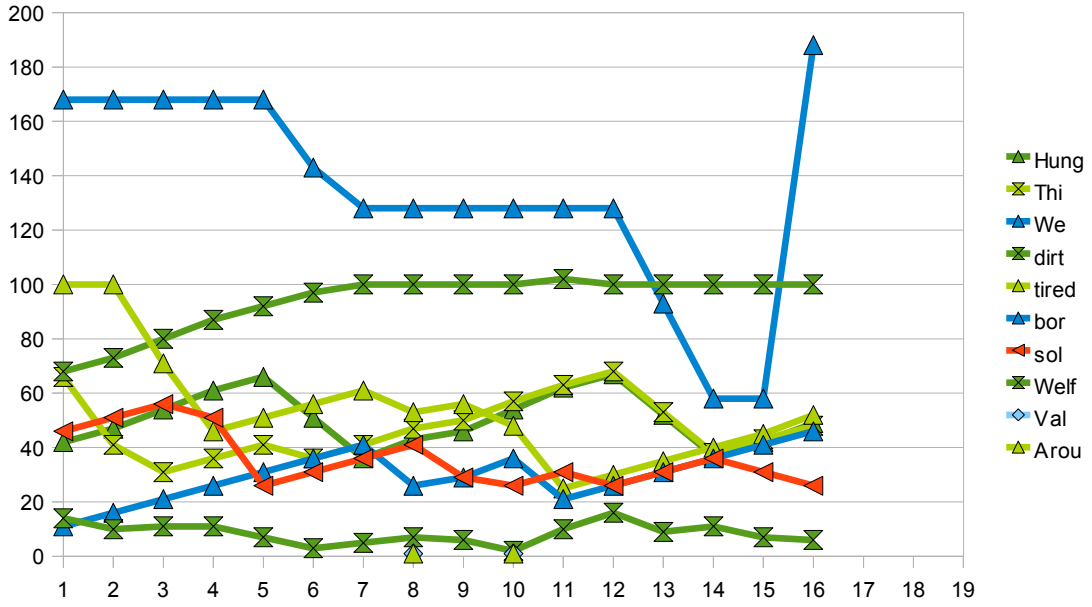


Ilustración 34: Ejecución actor 1 Mike

En la gráfica del actor Mike se aprecia que el servidor le asigna el valor inicial máximo para el drive cansancio (tiredness) y un valor por encima del mínimo del personaje para el drive que controla la sed (thirst). Debido a la prioridad de las reglas, el actor satisface su necesidad de disminuir la sed y después utiliza un objeto de descanso para mantener los dos drives por debajo del umbral máximo definido en su perfil.

El valor máximo de todos los drives está establecido en 100 para el cómputo del drive felicidad (welfare) salvo la riqueza (wealth) que representa la capacidad que tiene el personaje para comprar diversos objetos en el escenario.

amy, 85 ejecuciones, valores tomados en las acciones 1,5,10,15...85

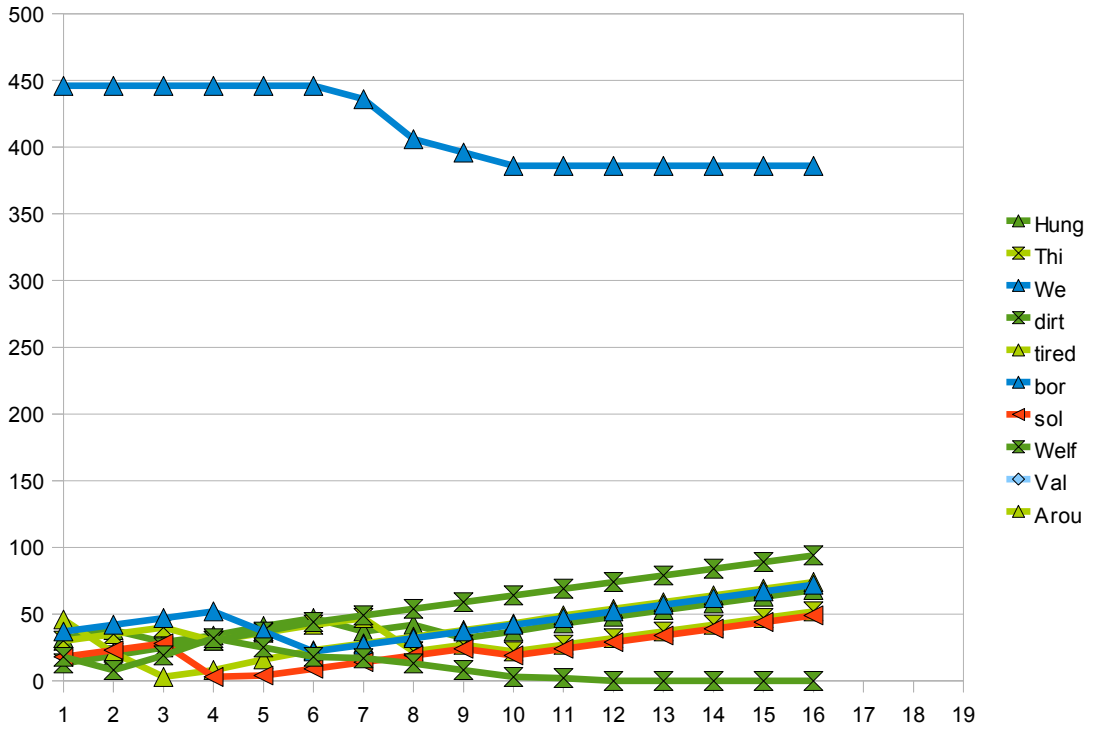


Ilustración 35: Ejecución actor 2 Amy

El segundo actor de las pruebas, Amy tiene unos valores máximos de sus drives configurados todos a unos niveles bastante bajos e iguales, entre 30 y 40. Esto provoca que llegado un momento dado, tiene muchos drives por encima de su valor mínimo que debe tratar de satisfacer y aunque consigue disminuir algunos, otros aumentan de forma lineal a lo largo del tiempo, siendo incapaz de satisfacer todas sus necesidades.

El problema de este actor reside en unos valores máximos muy bajos e iguales en todos sus drives, que debido al aumento de estos drives en cada turno simulando el paso del tiempo hacen que el actor no pueda satisfacerlos todos.

6.3 Resultados de la evaluación

Para medir los resultados del trabajo realizado, se realiza un seguimiento de dos actores controlados por la IA y se toman los valores de todos sus drives en cada momento, observando la evolución de los mismos.

Se observa que los actores intentan mantener los drives por debajo de sus límites máximos definidos en cada perfil pero debido a la cantidad de drives que intervienen, al crecimiento progresivo en cada turno y los conflictos que se producen con otros actores del escenario al coincidir en un mismo punto del escenario para usar el mismo objeto, algunos actores no consiguen reducir de forma inmediata los drives que están por encima de sus máximos definidos. Es necesario incluir un mayor número de objetos en el escenario que permitan interactuar a más actores de forma simultánea y ajustar el grado en que cada objeto alivia una necesidad del actor.

Los actores activan las reglas para satisfacer sus necesidades de forma correcta, atendiendo a las prioridades de cada regla y sus niveles máximos definidos en su perfil pero si esos niveles máximos no están bien balanceados, los actores no son capaces de satisfacer todas sus necesidades.

Se simula el paso del tiempo desde el servidor, introduciendo una mayor duración de ciertas acciones basada en el número de turnos que la acción puede durar. Además el servidor aplica un desgaste en los actores en cada turno, añadiendo cansancio y empeorando otros drives para obligar a los actores de esta forma a realizar diversas acciones que mejoren su situación.

7 Manual de usuario

7.1 Requisitos

La aplicación está diseñada para cualquier sistema con una versión de compilador compatible con GCC y que utilice hilos o threads POSIX y sockets BSD por lo que solo es necesario un sistema con el compilador apropiado y las bibliotecas de desarrollo.

Para la compilación en entornos *Windows* se requiere el compilador *MinGW* correctamente instalado y utilizar su implementación de *GNU Make*. En *Linux* se necesita el compilador gcc y otras utilidades como make. Para instalar todo lo necesario en sistemas basados en *Debian* como *Ubuntu* y sus derivados, instalar el paquete "build-essential". Para otras distribuciones, consultar la documentación correspondiente.

Se puede compilar directamente el código del cd o se puede descargar la última versión del código desde el repositorio. Para descargar la última versión se necesita tener instalado un cliente *SVN*.

Si se quieren utilizar el script de ejecución se necesita tener instalado *xterm* y un *bash* compatible.

Para utilizar el cliente gráfico 3d se necesita tener instalado *Ogre3D* o en su defecto su SDK para compilar el proyecto con él y un interfaz de ventanas disponibles en el sistema para la visualización. El cliente gráfico no está aún portado para entornos *Windows*.

Se pueden utilizar los binarios que se entregan en el cd siempre que el sistema cuente con las mismas librerías con las que se compiló, si al ejecutar aparece un error de librerías, se debe recompilar el código.

7.2 Compilación

7.2.1 Compilación de todos los módulos

Copiar el código distribuido en el cd en un directorio cualquiera del equipo o descargar la última versión del repositorio con el siguiente comando:

```
svn checkout http://pruebaailivesvn.googlecode.com/svn/trunk/ <directorio>
```

Donde <directorio> es la ruta local donde se copiará todo el código fuente.

Para compilar el proyecto basta con ejecutar "Make" en el directorio raíz y éste se encargará de compilar todos los módulos del sistema actual que incluyen servidor, cliente manual, cliente IA y la interfaz GUI 3d.

El código del motor *CLIPS*, motor emocional y archivos comunes se compilan en el directorio "build" que ha de existir antes de iniciar la compilación.

Para limpiar los archivos generados hay un target en el archivo Makefile que se encarga de borrar los ejecutables compilados y los código objeto intermedios.

7.2.2 Compilación manual por componentes

Se han añadido opciones para compilar los distintos módulos de forma manual por separado.

Los módulos disponibles para compilar por separado son:

- **server** compila el servidor
- **client** compila el cliente IA
- **client-manual** compila el cliente manual
- **gui** compila la interfaz Gui 3d (Solo en Unix)

7.2.3 Configuración de Make

No es necesario editar el archivo Makefile para compilar el proyecto salvo si se quiere compilar en diferentes entornos. Para entornos Windows hay una línea específica para activar la correcta compilación bajo ese entorno, para el resto de plataformas Unix soportadas hay que comentar esa línea.

Si se desean cambiar los directorios u opciones de compilación, dentro del Makefile están declaradas las variables para ello. A continuación se muestra un fragmento de dicho Makefile y sus opciones:

```
#WINDOWS=1
OSTYPE=$(shell uname)
CC=gcc
CFLAGS = -Wall -lm

COMMON_DIR=common
CLIPS_DIR=$(COMMON_DIR)/clips
EENGINE_DIR=$(COMMON_DIR)/emotional_engine
SERVER_DIR=server-source
CLIENT_DIR=client-source
CLIENTM_DIR=clientmanual-source
GUI_DIR=client-gui
BUILD_DIR=build

COMMON_OBJS=$(BUILD_DIR)/buffer.o $(BUILD_DIR)/rinput.o
EENGINE_OBJS=$(BUILD_DIR)/ee_clips_adapter.o $
(BUILD_DIR)/emotional_engine.o
SERVER_SRCS=$(SERVER_DIR)/server.c $
(SERVER_DIR)/clientlist.c
CLIENT_SRCS=$(CLIENT_DIR)/client.c
CLIENTM_SRCS=$(CLIENTM_DIR)/client.c

ifndef WINDOWS
     GUI_DEFINES =
     GUI_LIBS = OGRE CEGUI-OGRE CEGUI OIS
     GUI_CXX = g++
     GUI_CXXFLAGS = $(shell pkg-config --cflags $
(GUI_LIBS)) $(GUI_DEFINES)
     GUI_LD = g++
     GUI_LDFLAGS = $(shell pkg-config --libs $(GUI_LIBS))
     GUI_FINCLUDES = $(GUI_DIR)/serverCommunication.cpp $
(GUI_DIR)/auxiliarStringFunctions.cpp
endif
```

El directorio “BUILD_DIR” debe existir antes de iniciar la compilación.

7.3 Configuración

La configuración para ejecutar el proyecto se realiza en el fichero “conf-ialive” situado en el directorio raíz. A continuación se detallan las opciones disponibles

Variables globales para todos los clientes y servidor:

rootdir=directorio raíz del proyecto, contiene los archivos necesarios para compilar y ejecutarlo
profilesdir= directorio donde se alojan todos los perfiles de personajes
serverPATH=directorio donde se encuentra el ejecutable del servidor
clientPATH=directorio del cliente
clientManualPATH=directorio del cliente manual
clientGUIPATH=directorio del cliente gráfico
ontology=nombre del fichero de ontología, debe estar en el directorio raíz de la aplicación.
port=puerto donde el servidor recibe peticiones
sizex=geometría x de la ventana xterm
sizey=geometría y de la ventana xterm
posx=incrementos x para colocar las venentas xterm
posy=incrementos y para colocar las ventanas xterm
cleanfiles=nombre del script para la limpieza de archivos
host=ip o hostname de la máquina a la que conectar los clientes

Variables para el servidor
initial_state=initial.state
server=./server

Variables para el cliente IA
client=ejecutable del cliente
stageC=escenario por defecto al que conectar

Variables del cliente manual
clientmanual=ejecutable del cliente manual
stageM=escenario por defecto al que conectar

Variables del cliente gráfico
clientgui=ejecutable del cliente gráfico

7.4 Ejecución

7.4.1 Ejecución utilizando script

Para lanzar el programa, tan sólo hay que ejecutar el script run.sh y unos parámetros determinados para lanzar los distintos módulos. Esta es la forma de uso y su explicación.

```
./run.sh -C [server|manual|client] -s -c [profile] -m
[profile] -g
```

La opción -C sirve para limpiar trazas y ficheros de estadísticas de los clientes y servidor, tiene tres opciones, server client y manual que lanzan a su vez el script de limpieza de cada uno de esos módulos.

```
Ej: ./run.sh -C server -C client
```

Esta orden limpiaría las trazas y ficheros de estadísticas del cliente y los estados que se generan en el servidor.

7.4.1.1 Clientes

La opción **-c** (profile) lanza un cliente de IA utilizando el perfil que se indique. Se debe pasar el nombre del perfil sin la extensión propia ".profile" y se lee del directorio configurado anteriormente bajo la opción profilesdir

```
Ej: ./run.sh -c mike
```

Este ejemplo lanzaría un cliente de inteligencia artificial usando el perfil mike.profile que se debe encontrar bajo el directorio configurado en el archivo "conf-ialive" y con las opciones del mismo, para realizar la conexión al servidor y puerto predeterminados.

Opción: **-m** (profile) opción equivalente a la anterior pero para lanzar un cliente manual especificando el perfil a cargar de la misma forma.

```
Ej: ./run.sh -c mike -m amy
```

Este ejemplo ejecuta un cliente de inteligencia artificial autónomo utilizando el perfil mike.profile y un cliente manual con las características del perfil amy.profile

7.4.1.2 Servidor

Opción **-s** se utiliza para ejecutar un servidor con las opciones del archivo "conf-ialive" Nótese que solo puede haber un servidor escuchando en el mismo puerto de una máquina aunque podría haber varios servidores utilizando diferentes puertos de escucha.

7.4.1.3 Gui

Opción **-g** activa el cliente gráfico 3D con las opciones especificadas en su archivo gui.ini que está en el mismo directorio

que el binario. Las opciones de ese archivo incluyen la ip o nombre del servidor al que conectarse, puerto y escenario que representará.

7.4.1.4 Ayuda

Opción **-h** muestra una pequeña ayuda con los parámetros disponibles. Si se ejecuta el script sin ningún parámetro también se muestra esta ayuda de uso.

7.4.1.5 Ejecución manual de los componentes

También se pueden ejecutar los componentes de forma manual, sin ningún script ejecutando los binarios correspondientes y pasando a mano los parámetros necesarios. En entornos *Windows* por el momento es la única forma de ejecutar los componentes:

Los clientes, tanto de IA como manual se lanzan de la siguiente manera

```
./client
Usage: client <ontology> <serverhost> <port>
[<actor>] [<stage>]
(default stage is DEFAULT_STAGE, default actor is
DEFAULT_ACTOR)
```

Los parámetros que se especifican en orden son la ruta a la ontología, ip o nombre del servidor, puerto al que conectarse y como opciones la ruta del perfil del actor a usar (de nuevo sin la extensión .profile) y el escenario al que conectarse.

Para lanzar el servidor manualmente se utilizan los parámetros: ruta de la ontología, estado inicial para generar el escenario y el puerto de escucha:

```
./server
Usage: server <ontology> <initial.state> <port>
```

Es importante conservar el orden de los parámetros como se especifica.

8 CONCLUSIONES

A continuación se exponen las conclusiones a las que se ha llegado durante la realización de este proyecto.

El modelo implementado de cliente-servidor facilita enormemente la programación de reglas y comportamientos de los clientes ya que cada cliente solamente debe tener en cuenta el estado que le envía el servidor en su turno y puede ejecutar cualquier regla con independencia del estado de los demás actores conectados. Esto también supone una desventaja a la hora de programar acciones en las que intervengan varios actores ya que en un mismo turno sólo un actor ejecuta realmente una acción.

A la hora de crear una nueva acción para los actores hay que estudiar detenidamente la prioridad de la regla para que no entre en conflicto con las demás, evitando que se active.

Se hecha en falta un método para poder depurar la ejecución de las reglas o emular una ejecución paso a paso y poder utilizar los comandos y facilidades que proporciona *CLIPS* para depurar.

Actualmente solo se puede comprobar el estado parando la ejecución y cargando en una instancia de *CLIPS* diferente el fichero de estado del cliente, la ontología y las reglas, además de cargar las instancias necesarias para la ejecución de las reglas. Si se pudieran introducir desde AI-LIVE los comandos de debug ayudaría mucho en el desarrollo de las reglas.

El sistema para simular el paso del tiempo funciona y está centralizado en el servidor, además se ofrece la posibilidad a los clientes de continuar realizando una acción que dura más de un turno o seleccionar otra distinta.

Aunque los actores son capaces de mantener satisfechas sus necesidades, dependen de que el escenario al que se conectan tenga todos los objetos necesarios para realizar las distintas acciones y si aumenta el número de actores conectados al mismo escenario, se requiere un mayor número de objetos ya que varios actores necesitan usar el mismo objeto en varios turnos. Los niveles máximos y mínimos de los actores deben elegirse con cuidado ya que crecen muy rápidamente con cada turno y acción realizada ya que el servidor penaliza todos los drives de los actores en cada turno, simulando el paso del tiempo y además cada acción y objeto empleados, conlleva a su vez una mejora de un drive y un

empeoramiento de otro, creciendo así muy rápidamente las necesidades de los actores.

Los actores presentan un problema a la hora de coger y dejar objetos y es que tienen definida la acción de coger un objeto en cuanto lo tienen a su alcance, después de localizarlo debido a una necesidad determinada. El actor deja el objeto en el escenario después de utilizarlo y podría darse el caso de volver a coger el mismo objeto debido a la prioridad de las reglas. Para evitarlo se ha optado por coger solamente objetos que estén en la misma posición que el actor y dejarlos en una celda contigua después de usarlos.

El cliente gráfico proporciona gran cantidad de información, presentando una visión clara del escenario y la situación de los actores, además se muestra qué actividad están realizando en cada momento pero presenta algún inconveniente al usar cámaras fijas con escenarios pequeños como el baño o la tienda.

9 LÍNEAS FUTURAS/TRABAJOS

A continuación se proponen algunas ideas con las que mejorar o ampliar el proyecto.

- Mejorar la generación de ficheros de estado: actualmente se genera el fichero de estado de cada cliente y de cada escenario mediante una función de debug de *CLIPS* que provoca la aparición de demasiada información en la consola del servidor. Este proceso ralentiza bastante la ejecución del servidor.

- Definir una escala de tiempo y ajustar la duración de las acciones en turnos para simular el paso de los días en el sistema.

- Añadir más acciones y objetos con los que poder interactuar.

- Sería interesante permitir acciones en las que intervenga más de un actor, como por ejemplo, utilizar un juego de mesa con otro actor.

- Generar nuevos escenarios distintos y añadir más objetos a la interfaz gráfica.

- Permitir hacer debug en la ejecución de las acciones, permitiendo la ejecución paso a paso y la introducción de comandos de debug propios de *CLIPS* durante la comunicación entre clientes y servidor.

- Mejorar el comportamiento de los actores para decidir cambiar de escenario buscando objetos en concreto.

- Generalizar las reglas para relacionar una necesidad del actor con un objeto del escenario y de esta forma reducir el número de reglas necesarias para controlar el comportamiento de los actores.

- Controlar la necesidad de un actor para cambiar de escenario y ofrecer varios distintos donde poder trabajar, comprar, divertirse, etc.

10 BIBLIOGRAFÍA

D. Borrajo, N. Juristo, V. Martínez y J. Pazos, Inteligencia Artificial. Métodos y Técnicas, Centro de Estudios Universitarios Ramón Areces, Madrid, 1993

Castronova, E. Synthetic Worlds: The Business and Culture of Online Games, University of Chicago Press, 2005.

Laird, J. E. and van Lent, M. 1999. Developing an Artificial Intelligence Engine. In Proceedings of the Game Developers' Conference, San Jose, CA, 577-588.

R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 1985.

Rabin, S., Common Game AI Techniques. AI Game Programming Wisdom 2, AIWisdom.com, 2003.

Documentación obtenida de Proyectos de Fin de Carrera y Trabajos dirigidos previos:

[PÉREZ, 2006] Proyecto de Fin de Carrera: AI-LIVE.
Miguel Alfonso Pérez Bonomini.
Universidad Carlos III de Madrid. 2006.

[BENITO, 2007] Proyecto de Fin de Carrera: Necesidades básicas de actores en universos de realidad simulada.
Miguel Benito García.
Universidad Carlos III de Madrid. 2007.

[JIMÉNEZ, 2008] Proyecto de Fin de Carrera: Diseño e implementación de un modelo comunicativo emocional para agentes virtuales en el universo AI-LIVE.
Marta Jiménez Matarranz.
Universidad Carlos III de Madrid. 2008.

[UZQUIANO, 2008] Trabajo dirigido: Acceso manual al juego AI-LIVE.
Iván Uzquiano Mateo.
Universidad Carlos III de Madrid. 2008.

11 REFERENCIAS

[HALF_LIFE] Hl2spain.com. F.E.A.R. vs HALF- LIFE 2.
http://www.hl2spain.com/articulo.php?ar=14179

[IRRLICHT] http://irrlicht.sourceforge.net

[MULTIVERSE] http://www.multiverse.net

[Allegro] http://alleg.sourceforge.net

[KANEVA] http://en.wikipedia.org/wiki/Kaneva_Game_Platform

[SECOND_LIFE] Wikipedia - Second Life.
http://es.wikipedia.org/wiki/Second_Life

[SINGLES] Wikipedia - Singles: Flirt up your life.
http://en.wikipedia.org/wiki/Singles:_Flirt_Up_Your_Life

[SIMS] Wikipedia - Los Sims.
http://es.wikipedia.org/wiki/Los_Sims

[Los Sims] http://thesims.ea.com

[HISTORIA_VIDEOJUEGOS] Wikipedia - Historia de los videojuegos.
http://es.wikipedia.org/wiki/Historia_de_los_videojuegos

[NOVAMENTE] Novamente Artificial General Intelligence.
http://www.novamente.net

[GAMASUTRA] Interview: Novamente's Parrots And Advanced AI Progression http://www.gamasutra.com

[MMO]
http://en.wikipedia.org/wiki/Massively_multiplayer_online_game
http://www.mmorts.tk/
http://www.mmoreviews.com

[SCONS]
http://www.scons.org/wiki/SconsVsOtherBuildTools
http://www.scons.org/
http://www.scons.org/wiki/SconsVsOtherBuildTools

[BAZAAR]
http://doc.bazaar.canonical.com/

[Half-Life] Sdk de Half-Life
http://aigamedev.com/open/articles/halflife-sdk/

[Icub] an open source cognitive humanoid robotic platform
http://www.icub.org/index.php

[CMAKE] Why the KDE project switched to CMake -- and how
http://lwn.net/Articles/188693/
http://www.kde.org/
http://www.cmake.org/Wiki/CMake

[OGRE]
http://www.ogre3d.org/

[F.E.A.R] Assaulting F.E.A.R.'s AI: 29 Tricks to Arm Your Game
http://aigamedev.com/open/reviews/fear-ai/

12 Anexos

12.1 Manual de referencia

Este manual se encuentra en un documento aparte. Está disponible en la web del proyecto y es parte del repositorio SVN: http://code.google.com/p/pruebaailivesvn/source/browse/#svn/trunk/Documentacion