UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

Integración de emociones en agentes de JGOMAS

Trabajo Fin de Máster

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Peris Martínez, David

Tutores: Julian Inglada, Vicente Javier
Carrascosa Casamayor, Carlos

2014 - 2015

Resumen

Este trabajo consiste en la integración de emociones y personalidades en los agentes de JGOMAS, el cual es un entorno para desarrollar y ejecutar agentes sobre mundos 3D simulados. El modelado de las emociones se ha realizado según los modelos emocionales PAD y OCC para el diseño de 5 personalidades distintas. El objetivo final consiste en probar el funcionamiento de los agentes creados y comprobar si se comportan del modo previamente establecido. Con ello se quiere conseguir una mayor inmersión para el usuario que ejecuta el entorno de JGOMAS.

Palabras clave: emociones, agentes, JGOMAS, PAD.

Abstract

This work consists in embody of emotions and personalities of JGOMAS agents, which is an environment to develop and run intelligent agents over simulated 3D worlds. The emotions modeling has been developed as the PAD and OCC emotional models for the design of 5 different personalities. The final objective is to test the performance of the agents created and check if they behave as previously established. This is to be achived greater immersion for the user who executes the JGOMAS environment.

Keywords : emotions, agents, JGOMAS, PAD.

Tabla de contenidos

1 Introducción

1.1 Motivación

En el mundo de la Inteligencia Artificial existen una gran cantidad de herramientas o *toolkits*, las cuales se usan a menudo para modelar el comportamiento de unos agentes que interactúan entre sí.

A menudo se programan estos agentes de modo que tengan un fin útil, es decir, se les programa para que lleven a cabo una tarea compleja de forma eficiente, sin tener en cuenta las propias necesidades del agente. Un ejemplo de esto lo vemos en el algoritmo del enjambre de hormigas. Este algoritmo, que fue propuesto inicialmente por Marco Dorigo en 1992 (1), intenta emular el sistema que utilizan las hormigas para encontrar comida y llevarla a su hormiguero en el menor tiempo posible.

Los agentes de este sistema tan solo se preocupan en encontrar el camino más corto entre su hormiguero y la comida y por ello no se preocupan de cosas que puedan afectarles como el cansancio, el hambre o la sed.

Si en el mundo real tuviéramos agentes de este tipo, agentes mecánicos sin sensación alguna, resultaría en problema. Pues, ¿Qué pasaría si el hormiguero estuviera tan lejos de la comida, que solo para llegar hasta él las hormigas murieran de inanición? Sin duda esta única tarea conllevaría el exterminio de toda una colmena.

En este trabajo se traslada este dilema a un juego de un campo de batalla en el cual se enfrentan dos ejércitos igualados en fuerzas. Inicialmente ambos ejércitos se centran en curarse si están muy heridos, reponer su munición y, por encima de esto, cumplir sus objetivos. Cabe decir que aunque estos agentes estén malheridos, sin munición o en clara desventaja, nunca cesan en su empeño de alcanzar su objetivo, aunque esto signifique su propia muerte.

Por ello, en este trabajo me centro en 'humanizar' a los agentes, en darles emociones para ver cómo se comportarían en el campo de batalla a la par que valoren su propia supervivencia por encima del cumplimiento de los objetivos establecidos.

Con esto se pretende también sumergir en mayor medida al usuario en el juego dado que al tener emociones y actuar conforme a éstas, los agentes son más realistas.

1.2 Objetivos

Los objetivos a cumplir son los siguientes:

- Estudiar las distintas alternativas existentes para integrar emociones en agentes *software*.

- Definir distintas personalidades para los agentes de JGOMAS de acuerdo al modelo elegido con anterioridad.

- Diseñar esas personalidades mediante el modelo de emociones.

- Adaptar los modelos de agentes de JGOMAS para dar cabida a las personalidades anteriormente diseñadas.

- Comprobar mediante el desarrollo de diferentes pruebas el funcionamiento de esas personalidades para ver si los agentes actúan conforme a las acciones preestablecidas.

2 Agentes e Inteligencia Artificial

2.1 Introducción

La inteligencia artificial o IA puede considerarse como una de las disciplinas más nuevas, siendo curiosamente considerada a la vez como una gran desconocida y una de las que más interés profano despierta. Existen gran variedad de definiciones respecto a lo que es IA, pero podríamos resumirlas en que se trata de "desarrollar sistemas que piensen y actúen racionalmente" (2).

La IA fue introducida a la comunidad científica en 1950 por Alan Turing en su artículo "Maquinaria Computacional e Inteligencia." A pesar de que la investigación sobre el diseño y las capacidades de las computadoras comenzaron algún tiempo antes, no fue hasta que apareció el artículo de Turing que la idea de una máquina inteligente atrajo la atención de los científicos (3). Por otra parte, el término "inteligencia artificial" no fue acuñado hasta 1956, durante la conferencia de Darthmounth.

Una de las grandes razones por la cuales se realiza el estudio de la IA es él poder aprender más acerca de nosotros mismos y a diferencia de la psicología y de la filosofía que también centran su estudio de la inteligencia, la IA y sus esfuerzos por comprender este fenómeno están encaminados tanto a la construcción de entidades inteligentes como su comprensión (4).

Aparte del término IA que engloba cualquier comportamiento inteligente de un sistema, tenemos los agentes inteligentes que en el artículo "Agentes inteligentes, el siguiente paso en la Inteligencia Artificial" (2) los definen como un sistema de computación capaz de actuar de forma autónoma y flexible en un entorno. Entendiendo por flexible que sea:

- Reactivo: El agente es capaz de responder a cambios en su entorno.
- Pro-activo: También debe ser capaz de cumplir sus propios planes u objetivos.
- Social: Debe poder comunicarse con otros agentes utilizando un lenguaje de comunicación común.

Aparte de estas características que podríamos denominar como fundamentales de los agentes, otros autores como Franklin y Graesser (5), y Nwana (6),

mencionan una serie de características más o menos importantes que poseen los agentes:

- **Continuidad temporal**: Un agente es perpetuo y siempre desarrolla su función.

- **Autonomía**: El agente es capaz de adaptarse a los cambios del entorno.

- **Sociabilidad**: El agente puede comunicarse con otros agentes o entidades.

- **Racionalidad**: El agente siempre realiza lo que él considera como correcto.

- **Pro-actividad**: Es capaz de controlar sus objetivos a pesar de los cambios del entorno.

- **Adaptatividad**: Representa el aprendizaje que el agente puede realizar y en si puede cambiar su comportamiento en base a ese aprendizaje.

- **Movilidad**: Capacidad de un agente de transportarse a través de una red telemática.

- **Veracidad**: Asunción de que un agente no comunica información falsa a propósito.

- **Benevolencia**: Asunción de que un agente está dispuesto a ayudar a otros agentes, a menos de que esto entre en conflicto con sus objetivos.

No existe un consenso sobre el grado de importancia de cada una de estas características para un agente. No obstante se puede decir que estas propiedades son las que diferencian a los agentes inteligentes de meros programas (2).

2.2 Agentes BDI

Un agente BDI es un tipo particular de agente inteligente que sigue la arquitectura BDI (Creencias, Deseos e Intenciones). Este tipo de agentes tienen los siguientes componentes en su arquitectura:

- **Creencias**: Las creencias representan el estado de información del agente, lo que sabe de su entorno.

- **Deseos**: Los deseos representan el estado de motivación del agente. Representa los objetivos que al agente le gustaría cumplir.

- **Intenciones**: Las intenciones representan el estado deliberativo del agente, lo que el agente ha elegido hacer. En un sistema implementado significa que el agente ha empezado a ejecutar un plan. Los planes son secuencias de acciones que el agente puede utilizar para alcanzar una o más de sus intenciones.

- **Eventos**: Son disparadores de la actividad reactiva del agente. Un evento puede actualizar creencias, accionar planes o cambiar objetivos. Los eventos pueden ser generados externamente y recibidos mediante sensores o sistemas integrados (7).

Esta arquitectura tiene importantes ventajas respecto a otras arquitecturas de agentes (arquitecturas basadas en lógica, reactivas, etc.). Esto es así dado que ha podido reflejar el proceso de razonamiento humano de manera efectiva, dando lugar a robots robustamente filosóficos (8).

No obstante, el modelo BDI posee dos limitaciones conocidas. La primera es que carece de capacidad para el aprendizaje, la segunda es que carece de una funcionalidad multi-agente explícita. Ambas carencias constituyen un problema para lo que se conoce como aprendizaje en Sistemas Multi-Agente (9).

Hay ciertos estudios que han intentado aproximar la arquitectura BDI al proceso emocional humano, sin embargo estos estudios resultaron confusos en sus definiciones, y a veces lo que algunos afirmaban como verdadero era contradicho por los resultados de otros (8).

De acuerdo a Castelfranchi (10) los elementos básicos que constituyen las emociones son: las creencias, evaluaciones, objetivos, excitación, y la “tendencia tras la acción”. Las creencias son representaciones individuales del mundo que activan emociones con un nivel de excitación y motivan la acción.

En ese caso, las emociones vienen de la interpretación de los hechos y las sensaciones, lo cual dicho de otra forma puede ser llamado “reconocimiento de emociones”. Por otra parte, el individualismo tiende a evitar o tal vez a perseguir algún objetivo si este da lugar a un estado emocional deseable. En este sentido las emociones en sí mismas pueden considerarse como objetivos.

Un inconveniente importante que destaca Castelfranchi (10) es la diferencia entre las dos formas de apreciación de eventos o evaluaciones: Por un lado esta las Adaptativas e irracionales, las cuales son orientaciones automáticas, intuitivas e inconscientes de lo que puede estar bien o mal, también llamadas primarias según la literatura. Por el otro lado están las declarativas o explicitas, que son evaluaciones basadas en el razonamiento, están estrechamente relacionadas con los objetivos y son conocidas también como secundarias.

2.3 JASON

En el área de los lenguajes de programación orientados a agentes, AgentSpeak se ha convertido en uno de los lenguajes abstractos basados en la arquitectura BDI más influyentes. El tipo de agentes programados con AgentSpeak se referencian algunas veces a sistemas de planificación reactivos.

Jason es el primer intérprete reconocido para AgentSpeak, que incluye además comunicación entre agentes. Si lo combinamos con Saci, por ejemplo, el sistema multi-agente Jason se puede distribuir de forma sencilla sobre una red.

El lenguaje interpretado por Jason es una extensión del lenguaje de programación abstracto AgentSpeak, el cual fue originalmente creado por Anand Rao. Otra característica importante en comparación con otros sistemas de agentes BDI es que Jason esta implementado en Java y está disponible como *Open Source,* distribuido además bajo la licencia GNU LGPL.

Además de interpretar el lenguaje AgentSpeak, Jason posee estas características (11):

- Posee una negación fuerte, por lo que ambas asunciones, de mundo cerrado y abierto son aceptadas.

- Posee manejadores en caso de que un plan falle.

- Permite anotaciones en creencias y en etiquetas de plan que pueden ser usadas mediante funciones de selección avanzadas.

- Ofrece Soporte para entornos de desarrollo (los cuales suelen estar programados en Java).

- Ofrece soporte para organizaciones de sistemas multi-agente y agentes que razonan sobre ellos, utilizando el modelo Moise+.

- Ofrece (en Java) funciones de selección completamente personalizables, funciones booleanas y una arquitectura de agentes en su conjunto.

- También incluye una librería de “acciones internas”, las cuales son esenciales.

- Además permite extender de forma sencilla el sistema mediante acciones internas definidas por el usuario, las cuales se programan en Java.

- Por último ofrece un IDE en la forma de un JEdit o plugin de Eclipse, el cual incluye una herramienta de ayuda para la depuración.

La sintaxis de Jason se basa en el modelo BDI, en el cual existen las creencias, los planes, los objetivos, etc. Para explicar la sintaxis voy a mostrar un programa de ejemplo escrito en el lenguaje que utiliza Jason:

```
//Creencias Iniciales

creencia1.

creencia2( 5 ).

//Planes

+plan : creencia2( 5 ) //Condición que debe cumplirse.

<-      //Hacer algo

        !accion_objetivo; //Le añado este objetivo al agente

.

//Acciones de objetivo

+!accion_objetivo : creencia1 //Condición para cumplir el objetivo.

<-

        -creencia1;

        ?creencia2( Valor );

        if( not creencia1 ){

                -+creencia2( Valor - 1 );

                +creencia1;

        }

.
```

Las creencias pueden ser tanto simples (sé que existe la creencia creencia1) como contenedoras de valores (sé que la creencia2 tiene valor 5). Estas se pueden crear tanto inicialmente (escribiendo simplemente su nombre seguido

del símbolo de terminación que en Jason es el punto '.') como dentro de planes o acciones poniendo el signo '+' delante de la creencia. Para consultar el valor de una determinada creencia se utiliza el símbolo '?' delante de la creencia, y entre paréntesis el nombre de la variable (la primera letra debe ser mayúscula) donde se va a almacenar el valor de la creencia. Si se quiere actualizar el valor de una creencia se escribe '-+' antes del nombre de la creencia y entre paréntesis el nuevo valor que tendrá la creencia. Por último para eliminar una determinada creencia se pone el signo '-' delante de ella.

Por su parte, los planes se definen inicialmente con el signo '+' seguido del nombre del plan más ':' y la condición que de cumplirse ejecutará dicho plan. En el ejemplo mostrado, para que el plan "plan" se ejecute, la creencia "creencia2" debe alcanzar el valor '3'.

Las acciones de objetivo a diferencia de los planes se ejecutan dentro de otros planes o acciones poniendo el símbolo '!' delante del nombre de la acción, esto significa que se le añade ese objetivo al agente, que debe de cumplir. Una vez se cumple el objetivo, se activa la acción (12).

3 JGOMAS

3.1 Introducción

JGOMAS (*Game Oriented Multi-Agent System, based on JADE and Jason*) es una plataforma de prueba para estudiar la integración completa entre SMA y RV (Realidad Virtual), permitiendo también ser usada como una herramienta de validación de SMA.

Como simulación del entorno en el SMA desarrollado se ha elegido un juego del tipo capturar la bandera. En esta clase de juegos, dos equipos (rojo y azul, aliados y eje) deben competir para capturar la bandera del oponente. Esta modalidad de juego ha llegado a ser un estándar en casi todos los juegos multi-jugador que han aparecido desde Quake.

Es muy fácil e intuitivo aplicar SMA a este tipo de juegos, porque cada soldado puede ser visto como un agente. Además, los agentes de un equipo deben cooperar entre ellos para conseguir el objetivo del equipo, compitiendo contra el otro equipo. En esta versión de capturar la bandera, los dos equipos participantes son los aliados y el eje.

Los agentes aliados deben ir a la base del eje, capturar la bandera y volver a su base, en cuyo caso ganarán el juego. Por otro lado, los agentes del eje deben defender su bandera y, si es capturada, deben regresarla a la base. La partida tiene una duración máxima, transcurrida la cual, si la bandera no ha llegado a la base aliada, el equipo del eje ganará el juego (13).

Este proyecto se ha basado en esta plataforma de juego dado que combina la sencillez y versatilidad del lenguaje JASON con un entorno virtual en 3D para visualizar mejor a los agentes.

3.2 Arquitectura

JGOMAS está compuesto principalmente de dos subsistemas. Por una parte, hay un Sistema Multi-Agente con dos clases diferentes de agentes ejecutándose.

Uno de estos agentes controla la lógica actual del juego, mientras que los otros pertenecen a uno de los dos equipos, y estarán jugando el juego. Realmente, este subsistema es una capa que funciona por encima de una plataforma de sistema multi-agente JADE, así puede tomar ventaja de todos los servicios que JADE provee.

Jason-JGOMAS, que es la versión actual de la plataforma JGOMAS, permite el uso de agentes Jason para formar los equipos de agentes participantes. Por otro lado, se ha desarrollado un visor gráfico (*Render Engine*) ad-hoc para visualizar un entorno virtual 3D. De acuerdo a los requerimientos propios de aplicaciones gráficas (alto coste computacional por cortos períodos), este visor gráfico ha sido diseñado como un módulo externo (y no como un agente) (13).

3.2.1 Tipos de Agentes

Dentro de JGOMAS hay dos tipos de agentes principales:

1. **Agentes Internos**: Son los propios de la gestión de JGOMAS. Sus comportamientos están predefinidos, y el usuario no puede cambiarlos. Son agentes JADE, y existen los siguientes tipos:

    - Manager: Es un agente especial. Su objetivo principal es coordinar el juego actual. Además, debe contestar a peticiones del resto de agentes. Otra tarea de la que se encarga es de proveer de una interfaz para los visores gráficos. Por lo tanto, cualquier instancia del visor gráfico puede conectarse al juego actual y mostrar el entorno virtual 3D.

    - Packs: Hay tres tipos de packs o paquetes. Están los paquetes de vida o *medic packs* (los cuales curan a los agentes que los cogen), los paquetes de munición o *ammo packs* (los cuales recargan la munición de los agentes que los cogen) y los paquetes objetivos u *objective packs*, como puede ser la bandera a capturar. Todos estos agentes son creados dinámicamente a excepción de la bandera que permanece durante todo el juego (13).

2. **Agentes externos**: Representan a los jugadores del juego actual. Estos agentes poseen una serie de comportamientos predefinidos que el usuario puede modificar o añadir otros nuevos. Estos agentes se desarrollan en Jason y cada agente solo puede adoptar un rol durante la partida. Hay tres roles definidos, aunque el usuario puede definir nuevos, cada uno proveyendo un servicio único. Así, estos agentes, o *tropo agents*, se especializan en los siguientes roles:

    - Soldier: Es el tipo de agente más simple y sus funciones se limitan a disparar al enemigo en cuanto lo ven y en recuperar su propia vida y munición.

    - Medic: Este agente además de tener las funciones del *soldier*, debe suministrar paquetes de vida (*medic packs*) a los aliados heridos.

    - Fieldops: Trabaja de forma similar al *medic*, salvo que en lugar de paquetes de vida, reparte paquetes de munición (*ammo packs*).

Un dominio como el de capturar la bandera permite de una forma sencilla y entretenida establecer un campo de pruebas para algoritmos y optimizaciones en cada agente, así como para estrategias cooperativas y competitivas entre los distintos equipos.

Los agentes externos se encuentran integrados en el entorno virtual, esto permite la interacción entre ellos (por medio de la percepción de compañeros / enemigos cercanos) lo que permite la cooperación o coordinación con compañeros del mismo equipo.

Además, al estar situados en este entorno virtual, deben tener en cuenta las características del terreno en el que se encuentren (dificultad de movimiento, paredes, etc.). Por otra parte, toda la comunicación que se realiza entre los agentes que componen la plataforma se realiza por medio del paso de mensajes según los protocolos establecidos por FIPA ACL (13).

3.2.2 Mapas

JGOMAS puede usar diferentes mapas para definir su entorno virtual. La mayoría de estos mapas son de tamaño 256 x 256, donde la posición de los agentes vendrá dada por su coordenada (x,y,z), donde la 'y' valdrá siempre 0 dado que en los mapas suministrados la altura es nula.

Cada agente tiene acceso parcial al mapa donde se desarrolla la partida, puesto que pese a tener acceso a la información estática del mismo, tan sólo puede percibir los objetos que están a cierta distancia, en otras palabras, ven lo que hay dentro de un cono delante de ellos (13).

Los mapas están formados por 4 ficheros, los cuales explicaré usando el primer mapa, que es en el que he hecho las pruebas del trabajo, y el segundo mapa como ejemplo:

- map_0x_cost.txt: Este fichero define las paredes del mapa por medio del uso de * para indicarlo. El contenido de este fichero es el siguiente:

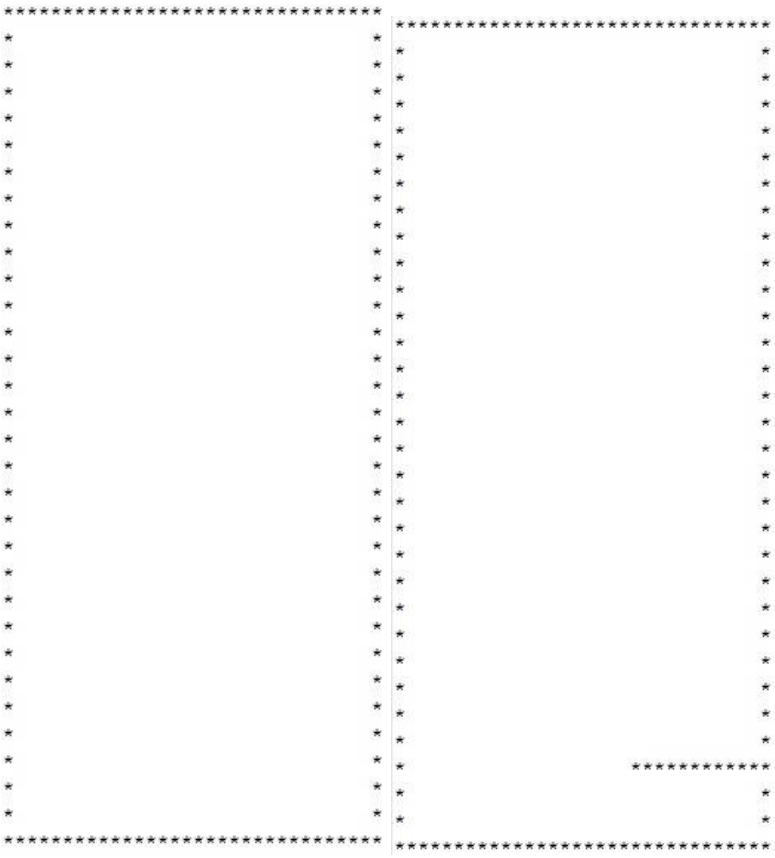

Ilustración 1: map_01_cost.txt y map_02_cost.txt.

- map_0x_terrain.bmp: Este fichero define el aspecto artístico del mapa.

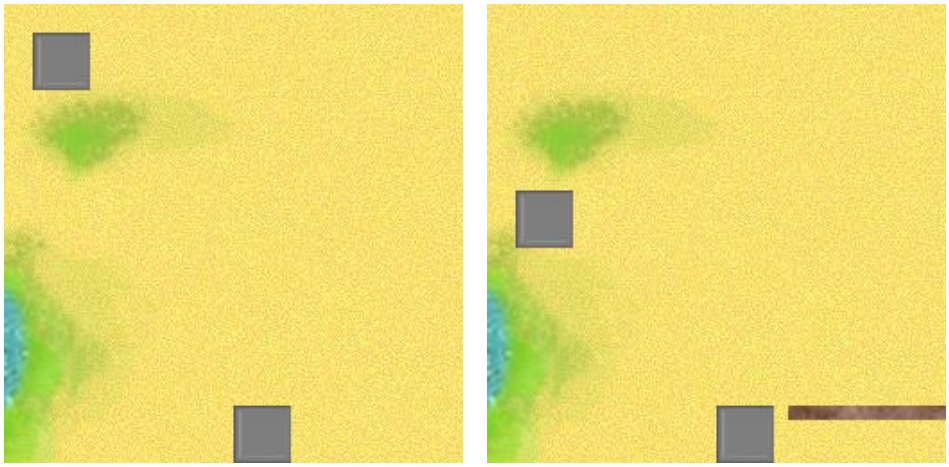

Ilustración 2: map_01_terrain.bmp y map_02_terrain.bmp.

- map_0x_cost.bmp: Este fichero define las paredes del mapa usando una imagen en blanco y negro, donde el negro representa la pared.

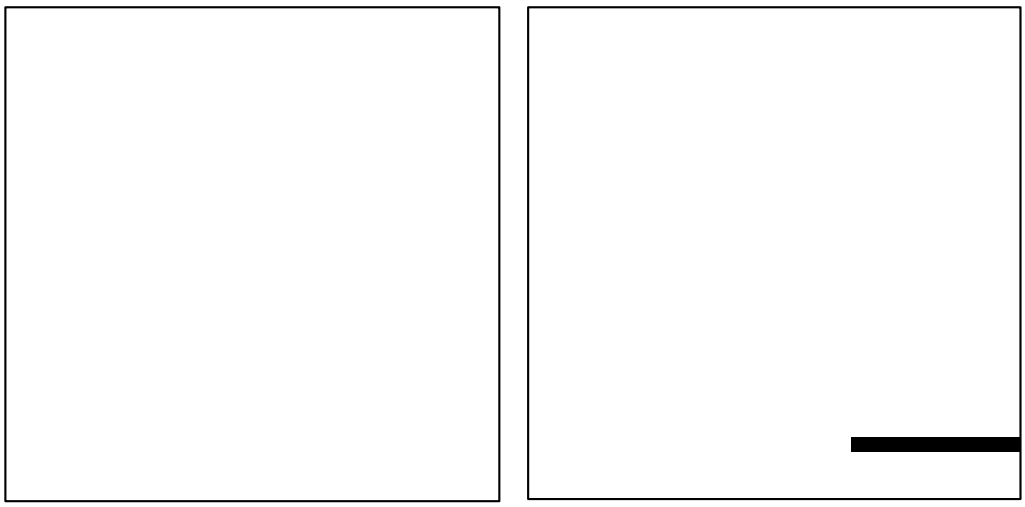

Ilustración 3: map_01_cost.bmp y map_02_cost.bmp.

- map_0x.txt: Este fichero contiene la definición de diferentes parámetros de configuración para le SMA y el visor gráfico. Entre ellos la posición de la bandera objetivo, el lugar de aparición de los soldados de los equipos aliados y eje; y el coste y nombre de los demás archivos del mapa.

```
[JADE]
JADE_OBJECTIVE: 23 30
JADE_SPAWN_ALLIED: 12 12 16 16
JADE_SPAWN_AXIS: 16 28 20 31
JADE_COST_MAP: 32 32 map_01_cost.txt
[JADE]

[RENDER]
RENDER_ART_MAP: 256 256 map_01_terrain.bmp
RENDER_COST_MAP: 32 32 map_01_cost.bmp
RENDER_HEIGHT_MAP: 32 32 map_01_heightmap.bmp
[RENDER]
```

Ilustración 4: map_01.txt.

```
[JADE]
JADE_OBJECTIVE: 23 30
JADE_SPAWN_ALLIED: 2 13 6 17
JADE_SPAWN_AXIS: 16 28 20 31
JADE_COST_MAP: 32 32 map_02_cost.txt
[JADE]

[RENDER]
RENDER_ART_MAP: 256 256 map_02_terrain.bmp
RENDER_COST_MAP: 32 32 map_02_cost.bmp
RENDER_HEIGHT_MAP: 32 32 map_02_heightmap.bmp
[RENDER]
```

Ilustración 5: map_02.txt.

3.3 Tareas

Una tarea es algo que un agente tiene que realizar en una posición concreta del entorno virtual en el que se encuentra. Existen distintos tipos de tarea, siendo las principales:

- TASK_GIVE_MEDIPACKS: El agente, si es un médico, debe generar *medic packs* en un lugar determinado (la posición del gente que lo ha solicitado).

- TASK_GIVE_AMMOPACKS: Un agente *fieldops* debe generar *ammo packs* en un lugar determinado (la posición del agente que ha solicitado munición).

- TASK_GET_OBJECTIVE: El agente del equipo 'Aliados' debe ir a la posición inicial de la bandera. Si ya posee la bandera, debe llevarla a su base.

- TASK_GOTO_POSITION: El agente debe ir a una posición.

Además de su tipo, una tarea tiene asociada el agente que lanzó la tarea, que puede ser el propio agente u otro que necesita algo, la posición donde se debe llevar a cabo, la prioridad de la tarea y opcionalmente un contenido adicional.

Cuando se lanza una tarea, se lanza la de prioridad más alta (se puede redefinir la prioridad de cada tarea). Por otra parte, las tareas las pone en ejecución el sistema mientras que el usuario tan solo puede añadir nuevas tareas a la lista de tareas activas del agente (13).

Para añadir una tarea se utiliza la acción: !add_task(task(Prioridad, Tipo, Agente, Posición, Contenido)).

3.4 Bucle de ejecución

Cada agente externo de JGOMAS ejecuta una máquina de estados como la que aparece en la ilustración 6. Los distintos estados son:

- STANDING: El agente no tiene ninguna tarea lanzada.

- GO_TO_TARGET: El agente ha lanzado una tarea y está moviéndose hacia la posición en la que debe realizarla.

- TARGET_REACHED: El agente ha alcanzado la posición en la que debe realizar la tarea y está realizando las acciones de esa tarea (13).

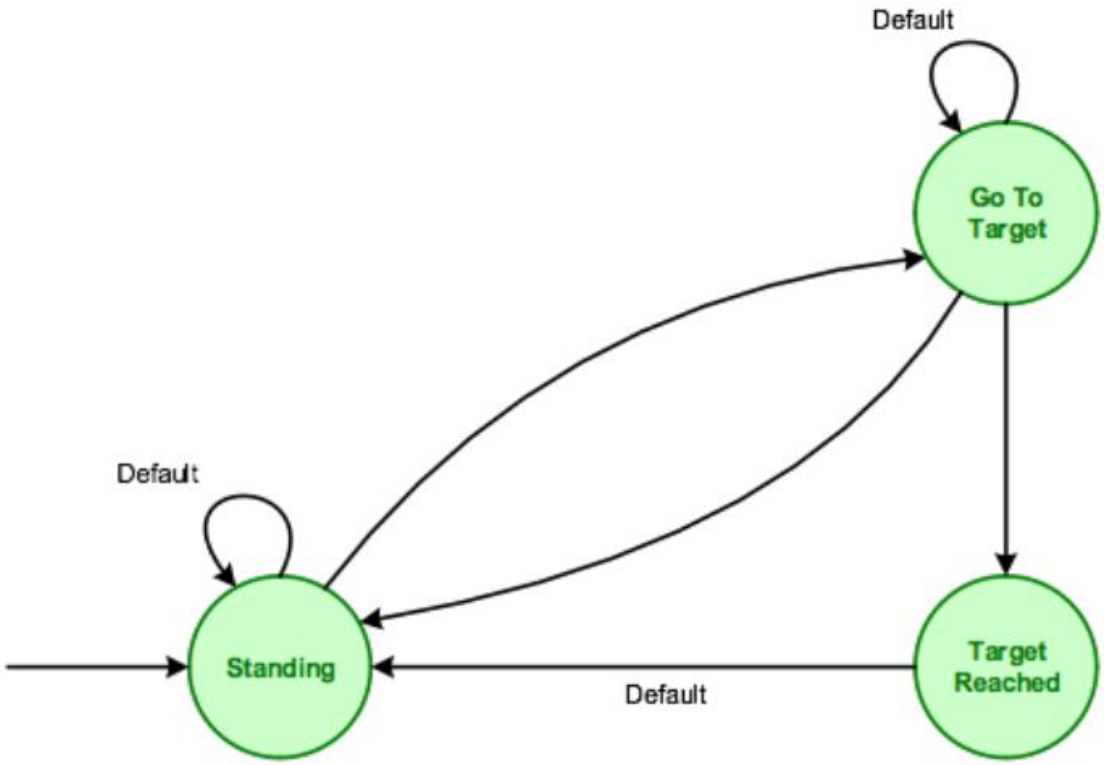

Ilustración 6: Maquina de estados que define el comportamiento de los agentes externos de JGOMAS.

3.5 Interfaz

Dentro del fichero 'jgomas.asl' se encuentran los comportamientos comunes a todos los agentes y que no se pueden modificar. Para modificar el comportamiento de cada agente, se debe modificar el archivo '.asl' de ese agente. Por ejemplo, para modificar el comportamiento del agente *Soldier* del equipo *Allied* se debe modificar el archivo "jason_Agent_ALLIED_SOLDIER.asl".

Las principales creencias de cada agente son:

- tasks(Task_list): Contiene la lista de tareas activas del agente.

- fovObjects(Lista_de_objetos): Contiene la lista de objetos que ve el agente en ese mismo momento. La estructura de un objeto es: [#, Equipo, Tipo, Angulo, Distancia al objeto, Vida del objeto, Posición del objeto].

- state(Estado_actual): Indica el estado del agente en su máquina de estados: *standing*, eligiendo que tarea hacer o esperando; *go_to_target*, dirigiéndose a su objetivo; *target_reached*, ha llegado al objetivo; *quit*, debe terminar la tarea.

- my_health(Vida): Guarda la salud del agente. El valor máximo e inicial es 100, si llega a 0 el agente se muere.

- my_ammo(Municion): Guarda la munición del agente: El valor inicial y máximo es 100.

- my_position(X, Y, Z): Almacena la última posición conocida del agente en coordenadas X Y Z.

Los planes que se pueden modificar de los agentes son los siguientes:

- !perform_look_action: Este objetivo se ejecuta cuando se ha mirado alrededor y se ha actualizado la lista de objetos *fovObjects(O)*.

- !perform_aim_action: Este objetivo se lanza si hay un enemigo para apuntar.

- !get_agent_to_aim: Este objetivo se lanza después de *!perform_look_action*, y se utiliza para saber si hay algún enemigo al que apuntar.

- !perform_no_ammo_action: Este objetivo se ejecuta cuando el agente dispara y no tiene munición.

- !perform_injury_action: Este objetivo se lanza cuando el agente es disparado.

- !performThresholdAction: Este objetivo se ejecuta cuando al agente le queda menos vida o munición que los umbrales definidos en las creencias *my_ammo_threshold(M)* y en *my_health_threshold(V)*. La implementación por defecto de este plan obliga al agente a pedir paquetes de munición y vida a otros agentes.

- !setup_priorities: Este objetivo se ejecuta en la inicialización del agente para definir las prioridades de las tareas.

- !update_targets: Este objetivo se invoca cuando el agente pasa a estado standing y debe elegir una nueva tarea entre las disponibles. Puede usarse para actualizar las tareas y sus prioridades. Es necesario implementar el plan asociado a la creación de este evento (13).

3.6 Comunicación entre agentes

Hay que destacar que un mismo agente asume un único rol durante la partida. Cada uno de estos roles tiene unas características y ofrece unos servicios básicos que pueden ser mejorados. La indicación del rol que adopta un agente se realiza durante la inicialización, mediante el proceso conocido como Registro. Aun así, un agente puede añadir nuevos servicios durante el desarrollo del juego.

3.6.1 Registro

Un rol debe registrar un servicio en el DF (Las páginas amarillas que utilizan los agentes para encontrar el nombre del agente con el cual se quieren comunicar) para que el resto de roles puedan solicitarlo. El registro se realiza mediante la función interna: *'.register("JGOMAS", "tipo")'*.

En el caso de los agentes de tipo *ALLIED* todos los agentes registran de forma transparente que pertenecen a dicho bando para que el resto de jugadores de su equipo lo puedan saber: *.register("TEAM", "ALLIED")*. Para los agentes del Eje o *AXIS* se haría de la misma forma pero cambiando el tipo por "AXIS".

Para registrar un agente de tipo soldado se registra el servicio "*backup_EQUIPO*", siendo "EQUIPO" *AXIS* o *ALLIED*: *.register("JGOMAS","backup_EQUIPO").*

Para registrar un agente de tipo *medic* se registraría el servicio "*medic_EQUIPO*", y para un agente de tipo *fieldops* se usaría *"fieldops_EQUIPO*" tal como he comentado antes.

Una vez un agente ha registrado un servicio, es posible que otro agente del mismo equipo consulte que agentes hay de su equipo que ofrecen un determinado servicio.

Para averiguar que agentes ofrecen un determinado servicio se utiliza la acción interna *".my_team(tipo, lista)"*. Esta acción devuelve la lista de agentes del equipo al que pertenece el agente que la invoca disponibles a través de las páginas amarillas (*DF*) que son del tipo escogido (excluyendo al propio agente) (13).

3.6.2 Coordinación entre agentes

JGOMAS dispone de algunos mecanismos que permiten la coordinación entre los agentes. Puede ser de dos tipos:

- Sin comunicación (implícita): Se consigue mediante la percepción del entorno por parte de un agente. Cuando un agente mira a su alrededor se lanza el objetivo !*perform_look_action*. Si se modifica este plan se puede decidir qué hacer en función de lo que el agente ve. Por ejemplo, al ver un grupo de enemigos se podría hacer que el agente se reagrupara con sus compañeros, que disparara o incluso que huyera del combate.

- Con comunicación (explicita): En este caso se utiliza el paso de mensajes mediante la acción interna *".send_msg_with_conversation_id(Receptor del mensaje, Performativa, Contenido, Id de la conversación)"*. Donde la performativa puede ser *tell*, *untell*, *achieve*, etc (13). Con una performativa "*tell*" se diría al agente al que se le envía el mensaje que ejecute un plan determinado, por otra parte con una performativa "*untell*" se impediría la ejecución de ese plan.

4 Emociones en Agentes Inteligentes

4.1 El modelo OCC

Un modelo basado en la evaluación que ha servido de base para la implementación de diversos modelos computacionales de emociones es el modelo cognitivo de emociones OCC. Este modelo fue desarrollado por los teóricos de emociones Ortony, Clore y Collins (de cuyas iniciales recibe su nombre).

Inicialmente, el modelo no fue orientado a la síntesis de emociones, sino para habilitar a los sistemas de IA para que razonaran acerca de las emociones, una capacidad especialmente útil para aplicaciones tales como orientadas al entendimiento del lenguaje natural y sistemas de diálogo.

Este modelo no usa emociones básicas, sino grupos de emociones de acuerdo a un esquema de condiciones cognitivas e inducidas. Se distinguen tres clases de emociones dependiendo del estímulo que ha provocado la emoción: Las inducidas por eventos, las inducidas por agentes y las inducidas por objetos. Utilizando esta estructura, se pueden especificar hasta 22 tipos de emociones.

Por otra parte, el modelo no es muy específico en cuanto a los detalles que conciernen a que valores se deben usar para los umbrales, o como las emociones interactúan, se mezclan y cambian su intensidad. Básicamente, el modelo OCC es un sistema basado en el conocimiento que se utiliza para generar diferentes tipos de emociones. Por lo tanto, no es el modelo más flexible. Además, se centra únicamente en como las percepciones influyen en las emociones pero no viceversa. Por otro lado no hay realimentación del sistema emocional al sistema cognitivo (14).

El modelo OCC provee a los científicos de un *framework* para el modelado y representación de las emociones. La mayoría de los sistemas afectivos contemporáneos están basados en este modelo. Además ha contribuido en gran medida a la apropiada representación de las emociones en los modelos afectivos (15).

4.2 El modelo PAD

Mehrabian propuso un modelo temperamental tridimensional para el análisis de los rasgos de la personalidad. Su análisis de rasgos emocionales (o temperamentos) se derivó a partir de descubrimientos relativos a los estados emocionales.

Los resultados obtenidos con el diferencial semántico y la comunicación no verbal permitieron la propuesta de un espacio de emociones tridimensional. Los tres ejes independientes de este espacio emocional (placer-desagrado, excitación-pasividad y dominación-sumisión) correspondieron a las evaluaciones, actividad y factores de potencia del diferencial semántico.

Los rasgos emocionales de un individuo se infieren de la media de sus estados emocionales junto con las muestras representativas de las situaciones cotidianas. Por lo tanto, Mehrabian propuso que los rasgos emocionales podían ser también descritos en términos de las dimensiones placer-desagrado, excitación-pasividad y dominación-sumisión.

Estudios recientes del modelo temperamental tridimensional (PAD) han usado con frecuencia las siguientes tres escalas:

- Rasgo Placer-Desagrado: Definido como el equilibrio, junto a las situaciones y por encima del tiempo, de los estados afectivos positivos sobre los negativos.

- Rasgo Excitación: Definido como mayor respuesta de la excitación y más lento acostumbramiento de la excitación a los estímulos inusuales, complejos o cambiantes. Cuanto más excitado se está, más rápido se percata uno de los cambios del entorno.

- Rasgo Dominación: Definido como los sentimientos habituales de control (más que la falta de control) sobre los momentos de la vida, los eventos u otros. Se podría entender como el deseo de hacer que todo permanezca como uno mismo lo desea.

Mehrabian se encontró con que el rasgo del placer y el rasgo de la dominación escalaban positiva y fuertemente correlacionados con la amabilidad y los factores de dominación, respectivamente, del modelo circumplejo. Por ello, dos líneas de investigación separadas sobre un periodo de 15 años identificaron la importancia del rasgo del placer (o amabilidad) y el rasgo de la dominación como fundamentales para la caracterización de la personalidad.

Además, los 5 grandes factores de la personalidad exhibían los perseguidos atributos del temperamento. La extroversión (placentera y dominante); la amabilidad (placentera y pasiva); La conciencia (placentera); La estabilidad emocional (placentera y sumisa); Y la sofisticación (placentera, excitable y dominante).

El espacio de temperamento tridimensional PAD fue definido por tres cercanamente independientes rasgos del temperamento. Varias dimensiones o medidas de la personalidad representan líneas rectas atravesando el punto de intersección de los tres ejes.

Los tres ejes fueron dicotomizados (divididos en dos) para descubrir varios tipos de temperamento: placer (+P) frente al desagrado (-P); la excitación (+A) frente a la pasividad (-A) y la dominación (+D) frente a la sumisión (-D). Los 8 tipos resultantes se etiquetan de la siguiente forma (16):

(+P+A+D) = Exuberante	contra	(-P-A-D) = Aburrido
(+P+A-D) = Dependiente	contra	(-P-A+D) = Desdeñoso
(+P-A+D) = Relajado	contra	(-P+A-D) = Ansioso
(+P-A-D) = Dócil	contra	(-P+A+D) = Hostil

Ilustración 7: octantes anímicos del espacio del PAD.

Por supuesto, estas etiquetas se pueden modificar y adaptar a otro tipo de estados como pueden ser el estado miedoso, el estado de pánico, el estado de optimista, etc…

La implementación del espacio de ánimo del PAD utiliza rangos de ejes desde el -1.0 hasta el 1.0 para cada dimensión. Un octante emocional representa una descripción discreta de un estado de ánimo, el cual es definido por tres valores de rasgo, el placer, la excitación y la dominación.

Por ejemplo, la descripción discreta de una persona será el estado “relajada”, si el valor del rasgo del placer es positivo, el valor del rasgo de la excitación es negativo y el rasgo de la dominación es positivo.

La fuerza del estado anímico actual se define por su distancia al punto cero del espacio del PAD (algebraicamente: la normal del vector PAD). Si por ejemplo esa misma persona (o agente) tiene los valores P=0.25, A=-0.18 y D=0.12, su descripción discreta es de “ligeramente relajada”. Por ello un estado anímico representa un punto en el espacio PAD.

Por otra parte hay que considerar el aspecto de que el estado anímico de una persona (o agente) se vuelve más intenso conforme la persona experimenta más

situaciones que apoyan ese estado anímico. Por ejemplo, si el estado anímico de una persona se puede describir como “ligeramente ansiosa” y varios eventos provocan que esa persona experimente la emoción del miedo, el estado anímico de esa persona puede pasar de moderada a “completamente ansiosa”.

En la siguiente tabla se muestra un mapeo de las emociones del modelo OCC que afectan al espacio del PAD (17):

Emotion	P	A	D	Mood Octant
Admiration	0.5	0.3	-0.2	+P+A-D Dependent
Anger	-0.51	0.59	0.25	-P+A+D Hostile
Disliking	-0.4	0.2	0.1	-P+A+D Hostile
Disappointment	-0.3	0.1	-0.4	-P+A-D Anxious
Distress	-0.4	-0.2	-0.5	-P-A-D Bored
Fear	-0.64	0.60	-0.43	-P+A-D Anxious
FearsConfirmed	-0.5	-0.3	-0.7	-P-A-D Bored
Gloating	0.3	-0.3	-0.1	+P-A-D Docile
Gratification	0.6	0.5	0.4	+P+A+D Exuberant
Gratitude	0.4	0.2	-0.3	+P+A-D Dependent
HappyFor	0.4	0.2	0.2	+P+A+D Exuberant
Hate	-0.6	0.6	0.3	-P+A+D Hostile
Hope	0.2	0.2	-0.1	+P+A-D Dependent
Joy	0.4	0.2	0.1	+P+A+D Exuberant
Liking	0.40	0.16	-0.24	+P+A-D Dependent
Love	0.3	0.1	0.2	+P+A+D Exuberant
Pity	-0.4	-0.2	-0.5	-P-A-D Bored
Pride	0.4	0.3	0.3	+P+A+D Exuberant
Relief	0.2	-0.3	0.4	+P-A+D Relaxed
Remorse	-0.3	0.1	-0.6	-P+A-D Anxious
Reproach	-0.3	-0.1	0.4	-P-A+D Disdainful
Resentment	-0.2	-0.3	-0.2	-P-A-D Bored
Satisfaction	0.3	-0.2	0.4	+P-A+D Relaxed
Shame	-0.3	0.1	-0.6	-P+A-D Anxious

Ilustración 8: Mapeo de las emociones OCC en el espacio del PAD.

5 Introducción de emociones en los agentes de JGOMAS

5.1 Introducción

Este trabajo se ha centrado en la creación de emociones para los agentes de JGOMAS. Para ello se ha basado en el modelo PAD, el cual tiene tres componentes (*Pleasure*, *Arousal*, *Dominance*) (18). Se ha utilizado este modelo principalmente por su sencillez (los estados emocionales se pueden representar con tan solo tres variables) y porque, por ello es fácil de implementar en Jason y por ende en JGOMAS. Además, los estados anímicos están fuertemente ligados a las emociones obtenidas mediante el modelo OCC (como se puede observar en la ilustración 8), lo cual facilita la reactividad de los agentes (responder ante estímulos externos).

En esta parte del trabajo se explica cómo se han implementado varias personalidades en los agentes de JGOMAS, de modo que dependiendo de la personalidad que tenga asignada reaccione de forma distinta ante los mismos eventos. Además se realiza una estimación de cómo deberían de comportarse los agentes en distintas situaciones. Acto seguido se comprueba si esa estimación es correcta probando a los agentes en el entorno de JGOMAS y observando su ejecución. Por último se desarrolla una evaluación en base al comportamiento observado.

5.2 Definición de personalidades

Siguiendo lo anteriormente comentado, por un lado tendríamos las emociones del modelo PAD como pueden ser el miedo, la alegría, el sentir vergüenza o el deseo de permanecer aislado del resto del mundo. Por el otro lado tendríamos los estados anímicos del agente como pueden ser el estar contento, enfadado, nervioso o relajado.

Aunque estos modelos sirven para crear agentes que sientan emociones y respondan ante ellas, hace falta un componente crucial para que los agentes se comporten de un modo más realista y por ende la inmersión del usuario sea mayor. Este componente es el de la personalidad individual de cada agente.

Si todos los agentes reaccionaran exactamente de la misma forma ante los mismos estímulos no habría interés en presenciar sucesivas simulaciones de su ejecución, pues su comportamiento sería demasiado predecible. Un ejemplo de ello en el mundo real sería que todos los seres humanos tuvieran agorafobia (miedo a los espacios abiertos) de este modo sabríamos que siempre que un ser humano se encontrara de repente en un lugar espacioso, como un parque abierto, le entraría un ataque de pánico.

En el mundo real hay además personas que aunque tengan compartan un miedo a algo en particular, no reaccionan de la misma forma frente a ese miedo. Por poner un ejemplo, dos personas les tienen miedo a las arañas, pero mientras que una de ellas se desmaya del susto, la otra tan solo trata de alejarse (pues a ambas les da miedo, pero no con la misma intensidad).

Como el entorno de JGOMAS se desarrolla en un campo de batalla, se considera que todos los agentes tienen miedo a la muerte en mayor o menor medida. Lo que va a diferenciar a unos agentes de otros va a ser la rapidez con la que se ponen nerviosos y deprimidos y contentos y relajados.

En el apartado 3, se ha explicado que JGOMAS tiene tres tipos de agentes externos que son el soldado, el médico y el repartidor de munición (*fieldops*). Para cada uno de estos agentes se han definido 5 personalidades distintas que son:

- **El agente por defecto** -> A este agente le afectan las emociones negativas y positivas en la misma medida. Se entiende por emociones positivas aquellas que aumentan el nivel del PAD del agente y le hacen alcanzar un estado anímico positivo (como es el estar contento). Las emociones negativas son, por el contrario, aquellas que pueden deprimir o provocar ansiedad en el agente.

- **El agente cobarde** -> Este tipo de agente se caracteriza por la rapidez con la que se pone asustado y nervioso. Las emociones negativas le afectan mucho y las positivas poco por lo que se recupera de forma muy lenta de los traumas (como puede ser el quedarse con muy poca vida o estar rodeado de enemigos).

- **El agente poco cobarde** -> Este agente es parecido al anterior salvo por el hecho de que las emociones negativas le afectan un poco menos y las positivas le afectan un poco más.

- **El agente valiente** -> Es el agente opuesto al agente cobarde y por ello las emociones positivas le elevan muy rápido el ánimo, mientras que las emociones negativas le reducen el ánimo muy despacio. Este tipo de agente es el que tarda más en retirarse del campo de batalla y vela menos por su propia vida.

- **El agente poco valiente** -> Igual al anterior, salvo que las emociones positivas le afectan un poco menos y las negativas un poco más. Este agente aguanta más tiempo en el campo de batalla antes de retirarse que el agente por defecto, pero menos que el agente valiente.

Como norma general, los diferentes estados anímicos en los que se encuentre un agente van a hacer que dispare y que pida ayuda más o menos a menudo, que se retire antes del campo de batalla o que pueda entrar en pánico (mientras que huir del campo de batalla se entiende como que el agente corre en dirección opuesta a la localización de los enemigos; el entrar en estado de pánico se entiende por correr hacia una posición aleatoria, que puede ser incluso hacia el propio enemigo). Además de estos comportamientos, un estado de ánimo determinado va a provocar que un agente de tipo *medic* o *fieldops* decida ayudar o no a un agente que le pide auxilio (es decir que valore más su propia vida que la de sus compañeros).

5.3 Modelado de las personalidades en el sistema PAD

Como se ha explicado en el apartado 4, el sistema PAD es un modelo tridimensional (se podría representar geométricamente con la forma de un cubo) en el cual los ejes están representados por las componentes 'P', 'A' y 'D'. En este trabajo se han discretizado estas componentes por lo que cada uno tan solo puede tener valores en el intervalo [-1, +1]. Se ha realizado de esta forma para controlar mejor las emociones, dado que por una parte el rango limita los valores mínimo y máximo de las componentes (se evita que las emociones tiendan al infinito y por ello se queden estancados en un estado positivo o negativo) y por otra permite diseñar mejor los umbrales (se tiene la certeza de que el valor mínimo es -1, el valor intermedio es 0 y el valor máximo es 1).

Cuando las componentes tienden a valores negativos se obtienen estados anímicos correspondientes al miedo, a la depresión, a la ansiedad, etc. Por el contrario, si tienden a valores positivos se obtienen estados anímicos como la alegría, la relajación, el optimismo, etc. Además, si las componentes tienden al valor intermedio (cero), se obtiene un estado anímico neutral en el que no se está ni contento ni deprimido.

Las componentes se pueden modificar de dos formas. La primera es que suceda un evento que provoque una emoción en el agente que a su vez modifique el valor del PAD de forma brusca (por ejemplo que se produzca de repente un terremoto). La segunda es que se mantenga ese evento o emoción en el tiempo, es decir que si el agente tras el susto inicial continúa sintiendo miedo, su valor PAD irá decreciendo gradualmente a lo largo del tiempo.

En este trabajo la componente 'P' está condicionada por el número de enemigos y aliados que tiene el agente a su alrededor. Si el agente está sólo, su componente 'P' crece a un ritmo lento (el nivel de variación está definido por una variable), si a su alrededor hay más camaradas que enemigos crece todavía más rápido, pero si en cambio percibe a más enemigos que compañeros, decrece en la misma medida (para la personalidad por defecto).

La componente 'A' está condicionada por la cantidad de vida que tenga en ese momento un determinado agente. Como el valor máximo que pueden tener los agentes de JGOMAS es 100 puntos de vida, si su vida se sitúa por encima de 50 puntos, la componente 'A' crecerá gradualmente con el tiempo y de forma opuesta si su vida se encuentra por debajo de los 50 puntos. Aparte, cada vez que el agente recibe daño su componente 'A' decrece bruscamente y si aumenta recupera vida.

La componente 'D' se modifica de la misma forma que la componente 'A' pero en función de la cantidad de munición que tenga el agente en un determinado momento. Esto influye en el hecho de que dependiendo de la cantidad de balas

que le queden al agente pueda decidir si desea disparar al enemigo o por el contrario guardárselas.

En el apartado 4 se explica que las componentes del PAD significan Placer, Excitación y Dominación. Como la componente 'P' está relacionada con el placer, se ha diseñado de forma que se modifique de modo que cuanto mayor sea el ejército del equipo del agente mayor placer tendrá este, dado que sabe que tienen superioridad numérica respecto del enemigo y de la misma manera se disgusta en el caso de que se dé la situación opuesta.

Por otro lado se ha realizado una ligera adaptación del modelo PAD enunciado por Mehrabian (16). En este trabajo, cuando la componente 'A' es positiva, el agente se encuentra en un estado de relajación mientras que si es negativa, este se encuentra excitado. Se ha realizado esta modificación porque en un campo de batalla es preferible mantener la calma a estar nervioso, pues ello puede conducir a cometer errores.

Una vez determinado como se va a modificar el valor del PAD a lo largo del tiempo en cada agente, se pasa a explicar las consecuencias de alcanzar uno u otro estado:

- **Exuberante / Alegre:** Este es el estado óptimo en el que se puede encontrar el agente. Sus tres componentes del PAD son positivas y la media de éstas es superior a +0.4. En este estado los soldados disparan siempre y los médicos y repartidores de munición siempre ofrecen apoyo a los que les llaman.

- **Tranquilo:** Estado en el que las componentes del PAD se sitúan en el intervalo:]-0.4 , +0.4[. En este estado los agentes disparan y ayudan a los otros el 70% de las veces. Se podría decir que es un estado clave, a partir del cual si recibe emociones positivas o negativas inclinará la balanza hacia ese lado.

- **Miedo:** Este estado se alcanza cuando al menos 2 de las componentes del PAD son negativas y la media de éstas es inferior a -0.4. En este estado los agentes disparan y ayudan tan sólo el 30% de las veces. Además si 2 de las componentes superan cierto umbral, en cuanto el agente divisa a un enemigo huye en dirección contraria.

- **Pánico:** Este es el peor estado emocional que puede alcanzar un agente. Para alcanzarlo primero se ha de alcanzar el estado anterior y si en ese momento las tres componentes superan un determinado umbral, el agente huye en una dirección aleatoria y no deja de huir hasta que se

recupere. Al ser una dirección aleatoria, puede ir hacia la dirección en la que se encuentra el enemigo. Además, durante este estado hace caso omiso a las peticiones de sus compañeros o a sus propias necesidades como puede ser el disparar al enemigo.

5.4 Resultado previsto

Si nos fijamos en las distintas personalidades que se han definido para los agentes, se dispone de 5 tipos de agentes distintos. La única diferencia entre ellos es como tienen fijado el valor de variación del PAD. A continuación se muestran unas gráficas sobre la variación del PAD en cada tipo de agente y en dos tipos de situaciones distintas, una favorable y otra desfavorable. En las gráficas se puede observar cómo debe de variar el PAD para cada agente dependiendo de sus parámetros.

Tabla 1: Situación favorable.

T	Enemigos	Aliados	Vida	Munición
0	0	0	100	100
1	0	0	100	100
2	0	1	100	100
3	1	1	100	100
4	0	1	100	100
5	0	1	100	100
6	0	1	100	100
7	0	1	100	100
8	0	2	100	100
9	0	2	100	100
10	0	2	100	100
11	0	2	100	100
12	0	2	100	100
13	0	2	100	100
14	0	2	100	100
15	0	2	100	100
16	0	2	85	85
17	0	2	85	85
18	0	4	85	85
19	0	4	85	85
20	0	4	85	85
21	0	4	85	85
22	0	4	55	55
23	0	4	55	55
24	0	4	55	55
25	0	4	55	55
26	0	4	70	70
27	0	4	100	100
28	0	4	100	100
29	0	4	100	100
30	0	4	100	100
31	0	4	100	100
32	0	4	100	100

La tabla 1 muestra una situación bastante favorable para los agentes, dado que el agente se encuentra rodeado de aliados y dispone una buena cantidad de vida y munición. Se puede observar que al principio el agente se encuentra solo, con la vida y munición completas. Conforme avanza la partida se le unen compañeros y a pesar de que gasta un poco de munición y vida, vuelve a recuperarse rápidamente.

Tabla 2: Situación desfavorable.

T	Enemigos	Aliados	Vida	Munición
0	0	0	100	100
1	0	0	100	100
2	0	1	100	100
3	1	0	100	100
4	0	0	95	90
5	0	0	95	90
6	0	0	95	90
7	2	0	95	80
8	2	0	95	70
9	2	0	95	50
10	2	0	95	50
11	2	0	95	50
12	2	0	85	50
13	2	0	95	50
14	2	0	95	50
15	2	0	95	50
16	2	0	95	50
17	4	1	75	30
18	4	2	50	40
19	4	2	30	40
20	4	2	30	40
21	4	2	30	40
22	8	2	10	40
23	8	2	10	10
24	8	2	10	10
25	8	2	10	10
26	8	2	10	10
27	8	2	10	10
28	8	2	10	10
29	8	2	10	10
30	8	2	10	10
31	8	2	10	10
32	8	2	10	10

Por el contrario, la tabla 2 muestra una situación muy desfavorable, pues el agente se encuentra rodeado de enemigos y ha perdido mucha vida y munición en muy poco tiempo. Se puede comprobar que mientras al principio de la partida se encuentra con un amigo y la vida y munición completas, conforme

avanza el juego, se rodea de enemigos y pierde una gran cantidad de vida y munición de forma drástica.

5.4.1 Personalidad por defecto

Todas las personalidades tienen la misma variación base del PAD para que los resultados sean uniformes. Este valor se usa para el cálculo del valor base (que es el mismo para todos los agentes). El valor que si varía dependiendo de la personalidad es el factor que multiplica al valor base positiva o negativamente. Por ejemplo para el agente por defecto este valor siempre es 1. En las siguientes ilustraciones se puede observar como varían las componentes del PAD a lo largo del tiempo:

Ilustración 9: Variación PAD del agente por defecto en una situación favorable.

Ilustración 10: Variación PAD del agente por defecto en una situación desfavorable.

El valor de la gráfica NENA (Numero de Enemigos menos el Numero de Aliados) muestra el número de enemigos menos el número de aliados (compañeros) presentes. Por ello, cuando el valor es negativo significa que hay más aliados que enemigos y cuando es positivo lo contrario.

5.4.2 Agentes poco cobarde y cobarde

El agente medio cobarde tiene un factor positivo de 0.7 y un factor negativo de 1.5. Esto significa que cuando el valor del PAD aumenta, lo hace a una razón del 70% del valor por defecto y cuando decrece lo hace a una razón del 150%. Por ello decrece más rápido que crece. El agente cobarde tiene por su parte un factor positivo de 0.5 y un factor negativo de 3. En las siguientes gráficas tan solo se

muestra la variación del agente cobarde para no excederse con el número de imágenes.

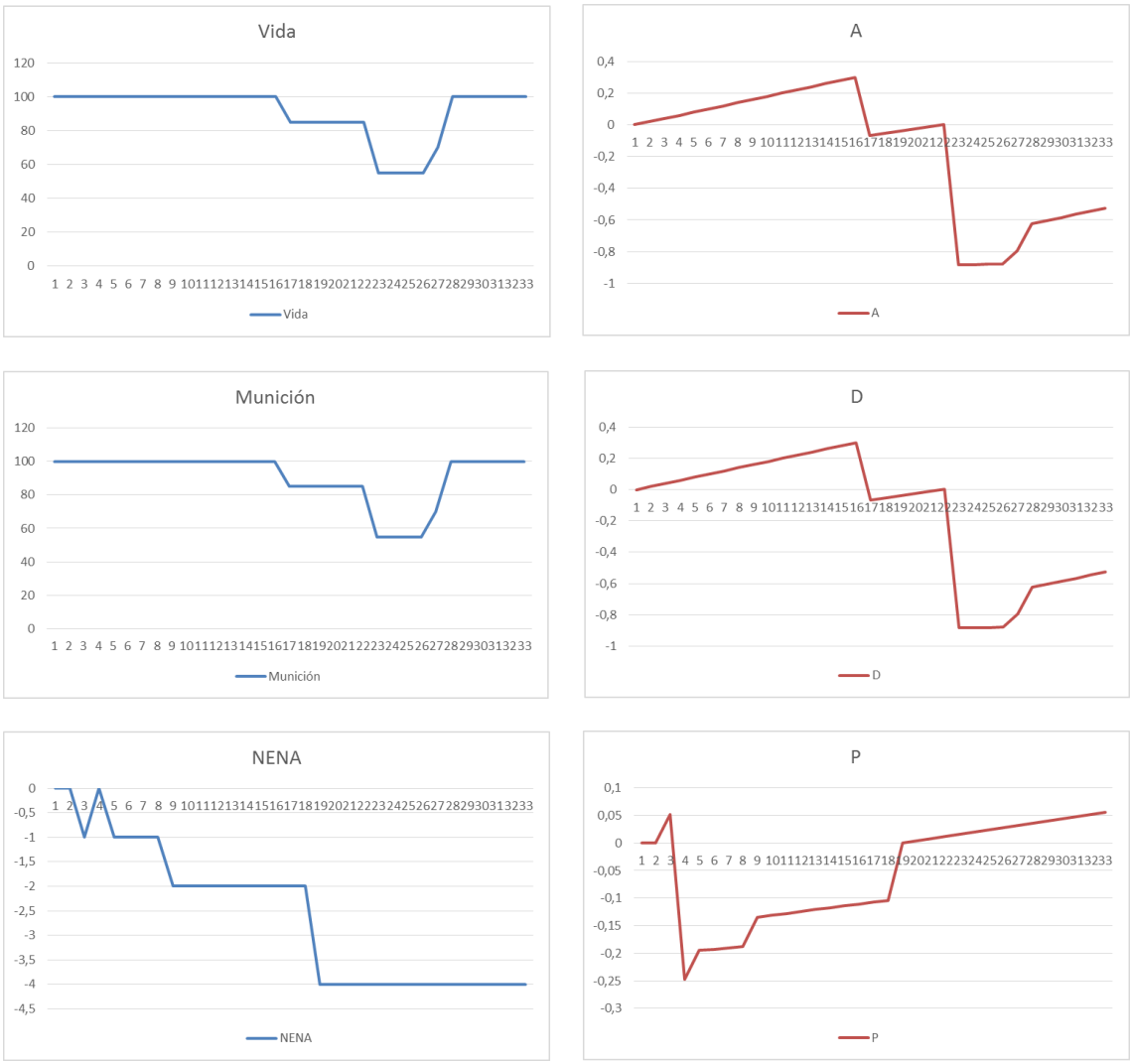

Ilustración 11: Variación PAD del agente cobarde en una situación favorable.

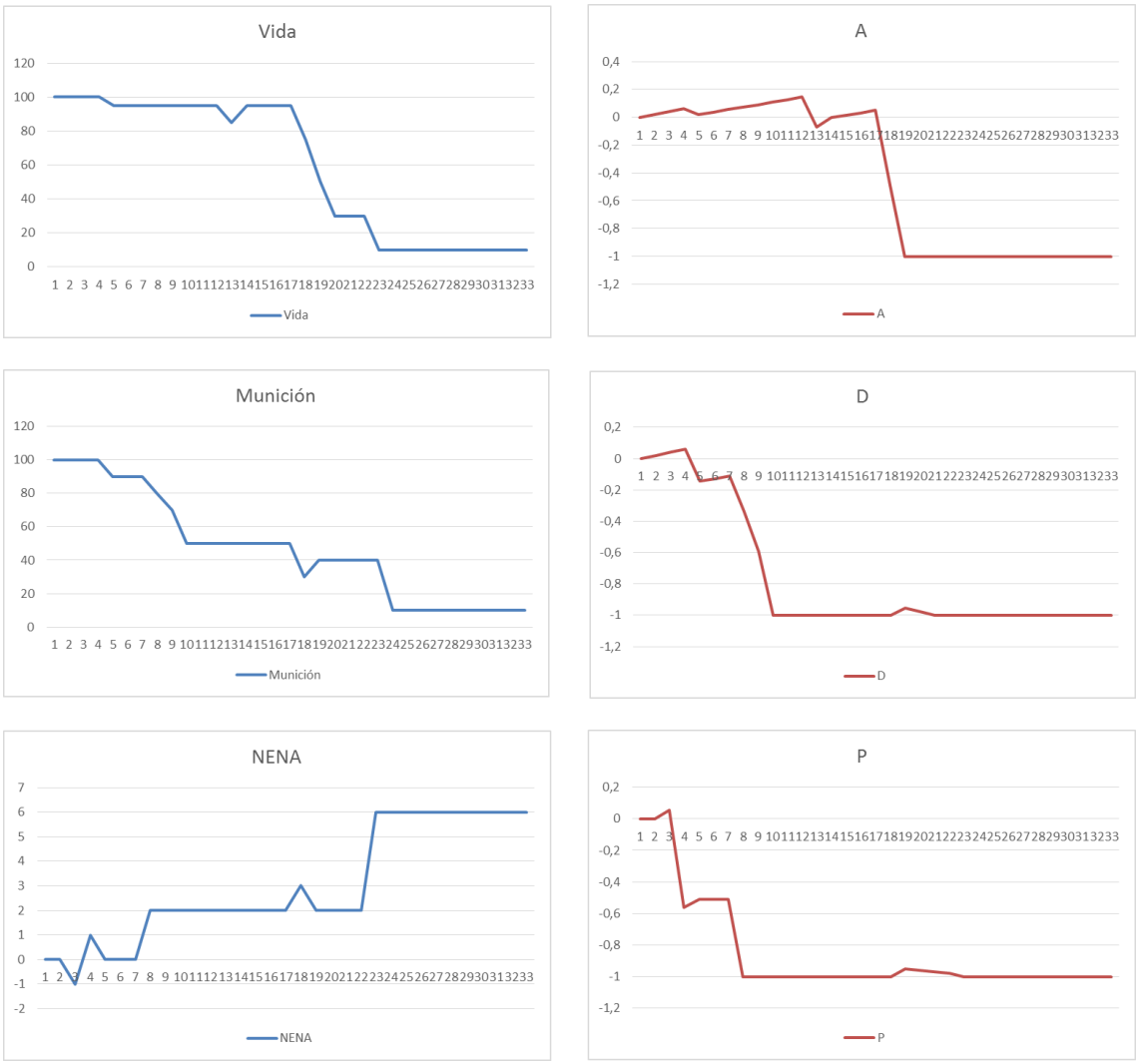

Ilustración 12: Variación PAD del agente cobarde en una situación desfavorable.

En las ilustraciones anteriores podemos observar que en cuanto el agente cobarde se encuentra con un contratiempo (por ejemplo perder vida) su nivel PAD baja bruscamente. Además, si se da la situación contraria su valor PAD se recupera de una forma mucho más lenta.

5.4.3 Agentes poco valiente y valiente

Estos agentes representan la forma opuesta de los agentes anteriores. El agente medio valiente escala con un factor negativo de 0.7 y positivo de 1.5. Por otra parte el agente valiente escala con un factor negativo de 0.5 y un factor positivo de 3.

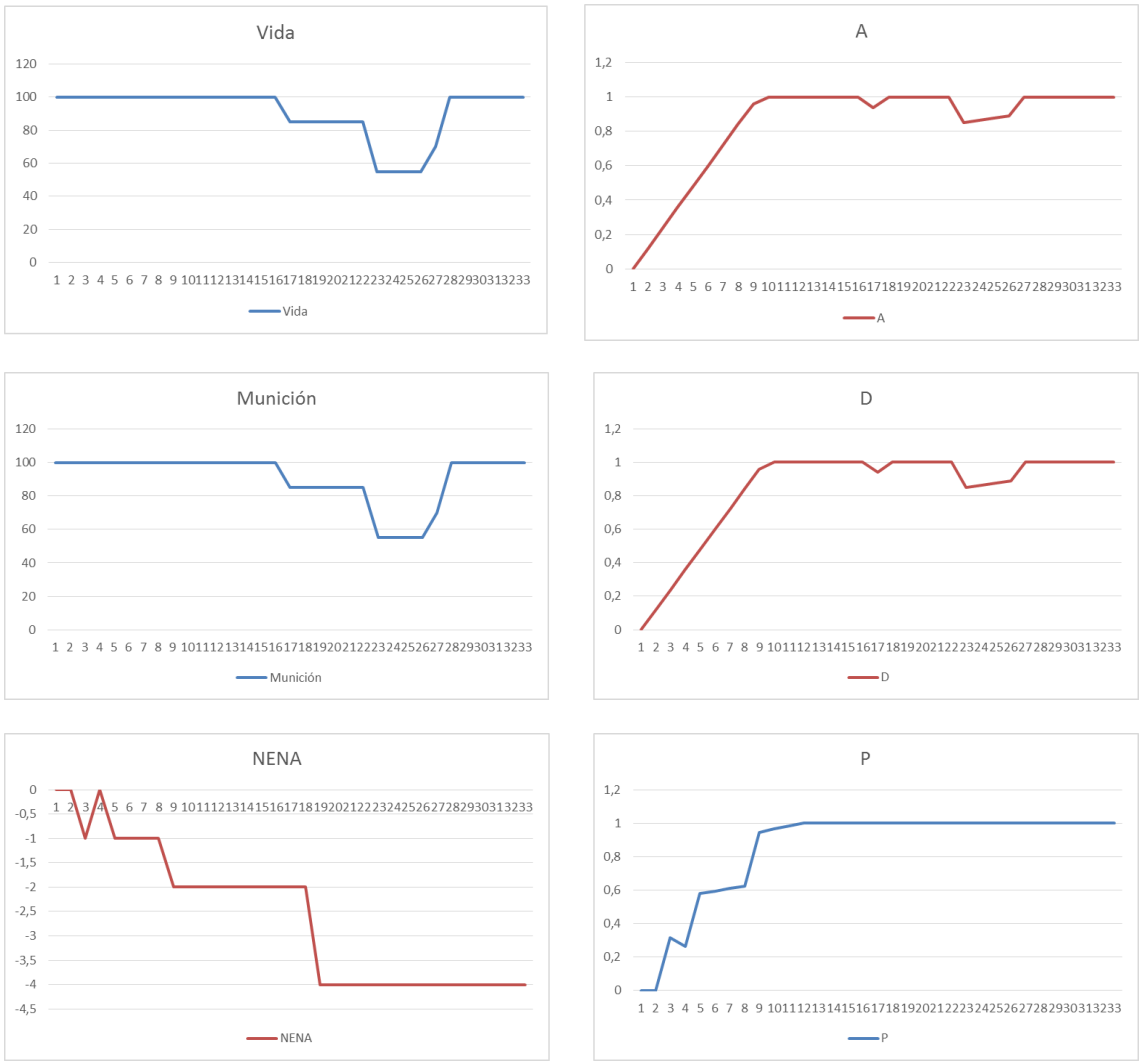

Ilustración 13: Variación PAD del agente valiente en una situación favorable.

Ilustración 14: Variación PAD del agente valiente en una situación desfavorable.

En este tipo de agente se puede observar que independientemente de la situación en la que se encuentre, siempre suele tener las tres componentes del PAD con un valor positivo. Como las situaciones desfavorables le afectan menos, siempre tarda más en sentir miedo o pánico, por lo que en raras ocasiones se retirará del campo de batalla.

5.5 Implementación

Dado que JGOMAS utiliza Jason para el modelado de agentes inteligentes y su código interno está escrito en Java (13), se ha tenido que adaptar este proyecto a estos lenguajes, con las ventajas y limitaciones que ello implica.

El código interno Java no se puede modificar con respecto al que trae JGOMAS por defecto. Esto implica que tan sólo se pueden utilizar las acciones internas predefinidas. Por otra parte, Jason tan sólo reconoce que un plan genera una determinada creencia si está definida explícitamente en el código interno de ese plan. Esto implica que en la ilustración 9 si el plan "plan_inicial" no tiene escrito en su código interno que genera la creencia "creencia1", cuando se añada el plan a la lista de objetivos el intérprete generará una excepción anunciando que nunca se podrá cumplir ese objetivo (a pesar de que en el código de Jason sí que haya un plan que genere esa creencia). Esto se debe a que por un lado están las creencias externas (Jason) y por otro las internas (Java) (12).

```
+plan_inicial
<-
    +creencia1;          //Añade una creencia
    !accion_objetivo;    //Añade un objetivo
.
+!accion_objetivo : creencia1   //Condición para cumplir el objetivo
<-
    //Se generará una excepción
    //No hay ningún plan interno que genere creencia1 -> ¡Excepción!
.
```

Ilustración 15: Objetivo cuya condición no se puede alcanzar nunca.

Como la mayoría del código Jason es el mismo para todos los agentes de un mismo equipo, se han modificado tan solo las creencias iniciales de cada agente y se ha compartido el archivo de tipo 'asl' que contiene la definición de los planes comunes a todos los agentes. Este archivo se llama "perfil_aliados.asl" para los agentes del equipo *ALLIED* y "perfil_eje.asl" para los agentes del eje.

Para la codificación se ha utilizado mayormente el editor de textos *notepad++* dado que se considera más versátil y práctico que el editor de textos que trae por defecto Jason (JEdit). A continuación se muestra un ejemplo con el agente por defecto del equipo *AXIS*, "jasonAgent_AXIS_DEFECTO.asl":

```
debug(4).

// Name of the manager
manager("Manager").

// Team of troop.
team("AXIS").
// Type of troop.
type("CLASS_SOLDIER").

// Value of "closeness" to the Flag, when patrolling in defense
patrollingRadius(64).

p_var(0.2). // mide el grado de variación del valor P de PAD

a_var(0.2). // mide el grado de variación del valor A de PAD

d_var(0.2). // mide el grado de variación del valor D de PAD

p_pos(1).
p_neg(1).

a_pos(1).
a_neg(1).

d_pos(1).
d_neg(1).


pad(0,0,0).


{ include("jgomas.asl") }
{ include("perfil_eje.asl")}
```

Ilustración 16: Código de jasonAgent_AXIS_DEFECTO.asl parte 1.

En la ilustración anterior se pueden ver todas las creencias iniciales que se crean al comienzo de la ejecución. La creencia *debug* se utiliza para depurar y su valor nos indica el nivel de verbosidad (*prints* por pantalla) que va a mostrar el agente. Cuanto más cercano del cero esté, mayor será la verbosidad. La creencia *manager* indica el *Manager* (agente interno) que gestiona su comportamiento.

Por el otro lado tenemos las creencias *team* y *type* que como su nombre indica guardan el equipo al que pertenece el agente y su tipo (*Soldier*, *Medic* o *AXIS*). La creencia patrollingRadius, propia del equipo eje, indica el radio de patrulla del agente. Cuanto más alto sea su valor, más lejos de la bandera patrullará.

Las creencias 'p_var', 'a_var' y 'd_var' miden el grado de variación base de las componentes del PAD. Son las variaciones bruscas que se producen cuando ocurre un evento (por ejemplo gastar munición). Por otra parte las variables

'p_pos', 'p_neg', 'a_pos', etc; marcan la variación de las componentes a lo largo del tiempo además de cambiar en los eventos. Por último, la creencia pad(P, A, D) va a guardar el valor inicial de las componentes del PAD y su modificación a lo largo del juego.

Las funciones *include* sirven para anexar el código de otros archivos que son comunes a todos los agentes, con lo que se evita la reiteración de instrucciones innecesarias. En los apartados siguientes se detallan los planes incluidos en los archivos "perfil_equipo.asl".

5.5.1 Plan get_agent_to_aim

Este plan es el encargado de calcular si hay algún enemigo a la vista. La versión por defecto inspecciona la lista de objetos contenida en la creencia *fovObjects* y si encuentra un enemigo actualiza la creencia *aimed* con el valor *true* para indicar que va a apuntar a ese enemigo (y a dispararle). En la siguiente ilustración se indica el cambio que se ha realizado en este plan:

```
if (Team == 200) {  // Only if I'm ALLIED

        //Compruebo el nivel de municion
        if( D > 0.4){
            ?debug(Mode); if (Mode<=2) { .println("Aiming an enemy. . .", MyTeam, " ",
            +aimed_agent(Object);
            -+aimed("true");
        }
        .random(V);
        if( D <= 0.4 & D >= -0.4 ){
            if(V > 0.3){
                ?debug(Mode); if (Mode<=2) { .println("Aiming an enemy. . .", MyTeam,
                +aimed_agent(Object);
                -+aimed("true");
            }
        }
        if( D <= -0.4 ){
            if(V > 0.7){
                ?debug(Mode); if (Mode<=2) { .println("Aiming an enemy. . .", MyTeam,
                +aimed_agent(Object);
                -+aimed("true");
            }
        }
}
```

Ilustración 17: implementación del plan get_agent_to_aim.

La variable *Team* indica el equipo del objeto observado por el agente. Si su valor es igual a 200 pertenece al equipo del eje y si es 100 pertenece a los aliados (es por ello que en el archivo "*perfil_eje.asl*" se comprueba si su valor es 100).

Después se comprueba el valor 'D' del PAD, si su valor es mayor que +0.4 siempre dispara al enemigo, en cambio si está entre -0.4 y +0.4 dispara el 70% de las veces (empieza a pensar si prefiere disparar o conservar munición), por otra parte si es inferior a -0.4 dispara tan solo el 30% de las veces.

5.5.2 Plan perform_look_action

Este es de los planes más importantes que trae por defecto JGOMAS dado que permite actuar al agente en función de los objetos que está observando el agente en ese momento. Este plan se ha modificado de modo que permite ver cuantos enemigos y aliados está observando el agente y guarde ese número en las creencias "enemigos(E)" y "aliados(A)".

```
.nth(1, Object, Team);
?my_formattedTeam(MyTeam);
?enemigos(Y);
?aliados(Z);
if (Team == 200) {  // Only if I'm ALLIED
    ?debug(Mode); if (Mode<=2) { .println("Encontrado enemigo. . .",
    -+enemigos(Y+1);

    if(miedo){
        //Si encuentra un enemigo huye.

        ?debug(Mode);
        if(Mode <= 4){.println("Enemigo encontrado"); }
        .nth(6, Object, pos(Xe, Ye, Ze));
        -+posEnemigo(Xe, Ye, Ze);
        if(Mode <= 4){.println("Enemigo encontrado2"); }
        +enemigoEncontrado;
    }

}
if (Team == 100){
    -+aliados(Z+1);
}
```

Ilustración 18: implementación del plan perform_look_action.

Además, en este plan se hace el primer paso para poder huir. En caso de que el agente tenga miedo (creencia 'miedo') si está viendo a un enemigo guarda su posición en la creencia "posEnemigo" y añade una creencia, "enemigoEncontrado", para saber que ha encontrado a un enemigo y debe huir en dirección contraria.

5.5.3 Plan checkMedicAction

Este plan lo utilizan los médicos para saber si deben ir a ayudar a quien les pide auxilio.

```
+!checkMedicAction
    <-
    ?debug(Mode);
   ?pad(P, A, D);
   Suma = (P + A + D)/3;
   .random(Al);
   if(Suma > 0.4){
       -+medicAction(on);// go to help
   }
   if(Suma <= 0.4 & Suma >= -0.4){
       if(Al > 0.3){
           -+medicAction(on);
       }else{ if (Mode<=4) {.println("NO AYUDO");}}
   }
   if(Suma < -0.4){
       if(Al > 0.7){
           -+medicAction(on);
       }else{ if (Mode<=4) {.println("NO AYUDO");}}
   }
.
```

Ilustración 19: Código del plan checkMedicAction.

Este plan funciona de la misma forma que el plan "*get_agent_to_aim*" salvo que en este plan se obtiene la media de las tres componentes del PAD y se guarda en la variable 'Suma'. Dependiendo del valor de esta variable ayudará siempre (actualizando la creencia '*medicAction*'), el 70% o el 30% de las veces.

5.5.4 Plan checkAmmoAction

Este plan lo utilizan solo los agentes *fieldops* para comprobar cuando tienen que dar munición a quien les pide ayuda. El plan funciona exactamente de la misma forma que el anterior. Dependiendo del valor de la media de las componentes del PAD, el agente recargará la munición del agente que le ha pedido auxilio el 30%, el 70% o el 100% de las veces.

```
+!checkAmmoAction
    <-
    ?pad(P, A, D);
    Suma = (P + A + D)/3;
    .random(Al);
    if(Suma > 0.4){
        -+fieldopsAction(on);// go to help
    }
    if(Suma <= 0.4 & Suma >= -0.4){
        if(Al > 0.3){
            -+fieldopsAction(on);
        }
    }
    if(Suma < -0.4){
        if(Al > 0.7){
            -+fieldopsAction(on);
        }
    }
.
```

Ilustración 20: Código del plan checkAmmoAction.

5.5.5 Plan performThresholdAction

En este plan se comprueban los umbrales de vida y munición para saber si el agente debe pedir ayuda a un agente *medic* o *fieldops*. Si la vida o munición baja de 90 puntos, se comprueba el valor de las componentes 'A' y 'D'. Si el valor de la componente 'A' es inferior a -0.2 pide vida. Si el valor de la componente 'D' es inferior a -0.2 pide munición.

```
if (Ar <= At) {
    if( D < -0.2 ){
        !pedirMunicion;
        if (Mode<=3) { .println("PIDO MUNI");}
    }
    else{if (Mode<=3) { .println("NO PIDO MUNI");}}
}
.random(RV);
if (Hr <= Ht) {
  if( A < -0.2 ){
      if (Mode<=3) { .println("PIDO VIDA");}
      !pedirVida;
    }
    else{if (Mode<=3) { .println("NO PIDO VIDA");}}
}
```

Ilustración 21: Código del plan performThresholdAction.

5.5.6 Plan updatePAD

Este es el único plan que se ejecuta de forma periódica. Cada 5 segundos este plan ejecuta el plan *"calculatePAD"* que ejecuta a su vez los planes *"calcEnemAli"*, *"calcVida"*, *"calcMunicion"* y *"corregirPAD"* para el cálculo del PAD. Después ejecuta el plan *"analyzePAD"* que se encarga de realizar las acciones pertinentes al estado emocional en el que se encuentre el agente.

```
+!updatePAD
   <-
   !calculatePAD;
   !analyzePAD;
   .at("now +5 s",{+!updatePAD});
   .
```

Ilustración 22: Código del plan updatePAD.

5.5.7 Plan calcEnemAli

Este es el plan que se utiliza para la actualización de la componente P del PAD. La variable 'PALFA' contiene la variación base del agente (0.2 por defecto), la variable 'PPOS' contiene el factor de escalado positivo y la variable 'PNEG' el factor de escalado negativo. Por otra parte está la variable 'NENA' que contiene la diferencia entre enemigos y aliados y la variable 'NUM' que calcula el valor entre la variable 'NENA' actual y la de la ejecución anterior.

Una vez se tienen estos valores se utiliza una fórmula para el cálculo del valor auxiliar:

➔ AUX = PALFA * (((NENA / (1+NE + NA)) * PALFA) + NUM)

Si AUX es positivo (hay más enemigos que aliados) se usa la siguiente fórmula para el valor de 'P' nuevo:

➔ Pnuevo = P - (AUX * PNEG)

En caso contrario:

➔ Pnuevo = P - (AUX * PPOS)

```
AUX = PALFA * (((NENA /(1+NE + NA)) * PALFA) + NUM);
//if(Mode<=4) {.println(AU);}
if(NUM == 0){
    P1 = P + ((PALFA/10) * PPOS);   //Si estoy solo me recupero a un ritmo constante y lento
}
else{
    if(AUX < 0){
        P1 = P - (AUX * PPOS);  //El valor acumulativo periódico debe ser pequeño.
    }
    else{
        P1 = P - (AUX * PNEG);
    }
}
-+last_nena(NENA);
-+pad(P1, A, D);
```

Ilustración 23: Código del plan calcEnemAli.

5.5.8 calcVida

Este plan calcula la componente 'A' del PAD. Para ello utiliza las variables 'ABETA', 'APOS', 'ANEG' que son las correspondientes a las explicadas en el plan anterior. Además utiliza la variable 'Dif' que calcula la diferencia entre la vida actual del agente y la vida en la ejecución anterior del plan.

Si la vida es mayor o igual a 30 puntos el valor auxiliar se calcula de la siguiente forma:

➔ AUX = ABETA * (((Vida - 50)/25) * ABETA + (Dif * 0.05))

En caso contrario se usa la siguiente fórmula:

➔ AUX = ABETA * (((Vida - 50)/50) * ABETA + (Dif * 0.05))

Con esto se diferencia un estado de alto riesgo (con menos de 30 puntos de vida el agente se encuentra cerca de la muerte), en el cual los cambios le afectan el doble de rápido, de un estado normal. Con esto se introduce un factor extra de ansiedad para simular que el agente se estresa porque sabe que su muerte se aproxima.

Para calcular el nuevo valor de 'A', si AUX es positivo se usa:

➔ Anuevo = Aactual + (AUX * APOS)

Si es negativo se usa esta fórmula:

➔ Anuevo = Aactual + (AUX * ANEG)

```
if(Vida < 30){
    AUX = ABETA * (((Vida - 50)/25) * ABETA + (Dif * 0.05));
}
else{
    AUX = ABETA * (((Vida - 50)/50) * ABETA + (Dif * 0.05));
}

if(AUX > 0){
    A1 = Av + (AUX * APOS); //El valor acumulativo periódico debe ser pequeño.
}
else{
    A1 = Av + (AUX * ANEG);
}
```

Ilustración 24: Código del plan calcVida.

5.5.9 calcMunicion

Este plan utiliza unas fórmulas muy parecidas al plan anterior para calcular el nuevo valor de 'D'. Las variables son muy parecidas salvo por que coge el valor de la munición en lugar del de la vida.

Para calcular el valor auxiliar, si la munición es inferior a 20 balas, utiliza la siguiente fórmula:

➔ AUX = D_var * (((Muni - 50)/25) * D_var + (Dif * 0.05))

En caso contrario utiliza:

➔ AUX = D_var * (((Muni - 50)/50) * D_var + (Dif * 0.05))

Como se puede observar lo único que varía es el umbral a partir del cual al agente le entra mayor ansiedad. El cálculo de la componente 'D' se hace de la misma forma que en el plan anterior como se puede observar en el código:

```
if(Muni < 20){
    AUX = D_var * (((Muni - 50)/25) * D_var + (Dif * 0.05));
}
else{
    AUX = D_var * (((Muni - 50)/50) * D_var + (Dif * 0.05));
}

if(AUX > 0){
    D1 = Dv + (AUX * DPOS); //El valor acumulativo periódico debe ser pequeño.
}
else{
    D1 = Dv + (AUX * DNEG);
}
```

Ilustración 25: Código del plan calcMunicion.

5.5.10 Plan analyzePad

Con este plan llegamos al análisis del valor del PAD para determinar el estado anímico en el que se encuentra el agente. Primero comprueba que las tres componentes del PAD sean inferiores a -0.5 y si es así entra en estado de pánico. En caso contrario comprueba si al menos una de las componentes es inferior a ese umbral y en caso afirmativo pasa al estado de miedo.

```
if( P < Umb & A < Umb & D < Umb ){
    +panico;
    !panico;
}
else{
    if( (P < Umb) | (D < Umb) | (A < Umb) ){
        +miedo;
        !huir;
    }
    else{
        if(miedo){
            -miedo;
            -enemigoEncontrado;
        }
    }

}
```

Ilustración 26: Código del plan analyzePad.

5.5.11 Plan huir

Este plan se activa cuando el agente tiene miedo (después del "*analyzePad*"). Una vez el agente ha avistado a un enemigo y comprueba que al menos dos de sus componentes del PAD son inferiores a -0.5 realiza la acción de huir que consiste en correr en la dirección contraria a la posición del enemigo. Para ello se utilizan vectores unitarios escalados para asegurar que el agente recorre siempre la misma distancia. Además nos aseguramos de que no se salgan de los bordes del mapa para que no se queden inmovilizados.

```
?my_position(X,Y,Z);
if(((P < Umb & A < Umb) | (D < Umb & P < Umb) | (D < Umb & A < Umb)) & miedo & enemigoEncontrado){

    ?posEnemigo(Xe, Ye, Ze);
    if(Mode<=4) {.println("HUYO");
    .println("X = ", X, " Xe = ", Xe, " Y= ", Y, " Ye= ", Ye, " Z= ", Z, " Ze= ", Ze );}

    Xv = (X - Xe); //Vector X
    Zv = (Z - Ze); //Vector Z
    //Haria falta usar vectores unitarios para que no se vayan demasiado lejos
    Modulo = math.sqrt(Xv * Xv + Zv * Zv);

    X2 = (X + 20*(Xv / Modulo));

    Z2 = (Z + 20*(Zv / Modulo));

    if(X2 > 247){ X2 = 247;}
    if(X2 < 8){ X2 = 8;}
    if(Z2 > 247){ Z2 = 247;}
    if(Z2 < 8){ Z2 = 8;}

    !add_task(task(2000, "TASK_GOTO_POSITION", Ag, pos(X2, Y, Z2), ""));
    -+state(standing);
    .println("MOVIENDOME A X = ", X, " X2 = ", X2, " Y= ", Y, " Z= ", Z, " Z2= ", Z2 );
    }
```

Ilustración 27: Código del plan huir.

El punto al que debe dirigirse el agente se ha calculado obteniendo primero el vector que señala la dirección contraria a la que se encuentra el enemigo (restando la posición del enemigo a la posición del agente) y sumando después este vector a la posición actual del agente.

La acción *"add_task(task(2000, "TASK_GOTO_POSITION", Ag, pos(X2, Y, Z2), ""))"* añade al agente la tarea de ir a la posición marcada por las coordenadas 'X2', 'Y' y 'Z2'. Esta tarea se activa al actualizar la creencia *'state(standing)'*.

5.5.12 Plan pánico

Como su nombre indica, este plan se activa cuando el agente ha entrado en estado de pánico y por lo tanto debe correr hacia una posición aleatoria. El agente deja de correr cuando alguna de sus componentes PAD supera el umbral.

```
if(panico){
    .my_name(Ag);
    R = 10;
    .random(EjeX);
    .random(EjeY);
    Dx = EjeX * 2 - 1;
    Dy = EjeY * 2 - 1;
    Radio = 40;
    .println("PANICO");
    ?my_position(Ax,By,Cz);
    X = Ax + Radio * Dx;
    Y = By;
    Z = Cz + Radio * Dy;
    if(Mode<=4) {.println(X," ",Y," ",Z);}
    ?pad(P,A,D);
    Umb = -0.1;

    if( P>Umb | A>Umb | D>Umb){
        -panico;
    }else{
        .my_name(Ag);
        !add_task(task(2000, "TASK_GOTO_POSITION", Ag, pos(X,Y,Z), ""));
        -+state(standing);
        if(Mode<=4) {.println("panico, voy a posicion X: ", X," Y: ",Y, " Z: ",Z );}

    }

}
```

Ilustración 28: Código del plan pánico

La función *.random(X)* devuelve un valor aleatorio entre el 0 y el 1. Esta función nos permite crear unas coordenadas aleatorias dentro de un radio cuyo centro es el propio agente (obteniendo el valor de los ejes 'X', 'Y' y 'Z'). Acto seguido, si el agente sigue en el estado de pánico (las componentes del PAD están por debajo del umbral), se añade una tarea para ir a la nueva posición calculada.

5.6 Pruebas

En este apartado se muestran unos ejemplos de ejecuciones del entorno JGOMAS con distintos tipos de agentes para comprobar si se comportan del modo en el que se ha establecido. Durante la ejecución hago diversas capturas de pantalla para comprobar cómo se va desarrollando el juego. La máquina sobre la que se han desarrollado las pruebas es un pc de escritorio con sistema operativo Windows 7, procesador AMD Phenom II X4 955 BE (con dos núcleos desbloqueados) y dos módulos de memoria ram Corsair Vengeance DDR3 de 1600 MHz. Todas las pruebas se han realizado sobre el mapa nº 1 de JGOMAS, con una velocidad de refresco de 125 milisegundos (se renderiza un fotograma cada 125 milisegundos) y con una duración máxima de partida de 10 minutos.

5.6.1 Primera prueba

En la primera prueba he creado dos equipos pequeños y balanceados. En ambos equipos hay 2 agentes *fieldops* con la personalidad por defecto, 2 agentes *medic* valientes y un agente *soldier* cobarde.

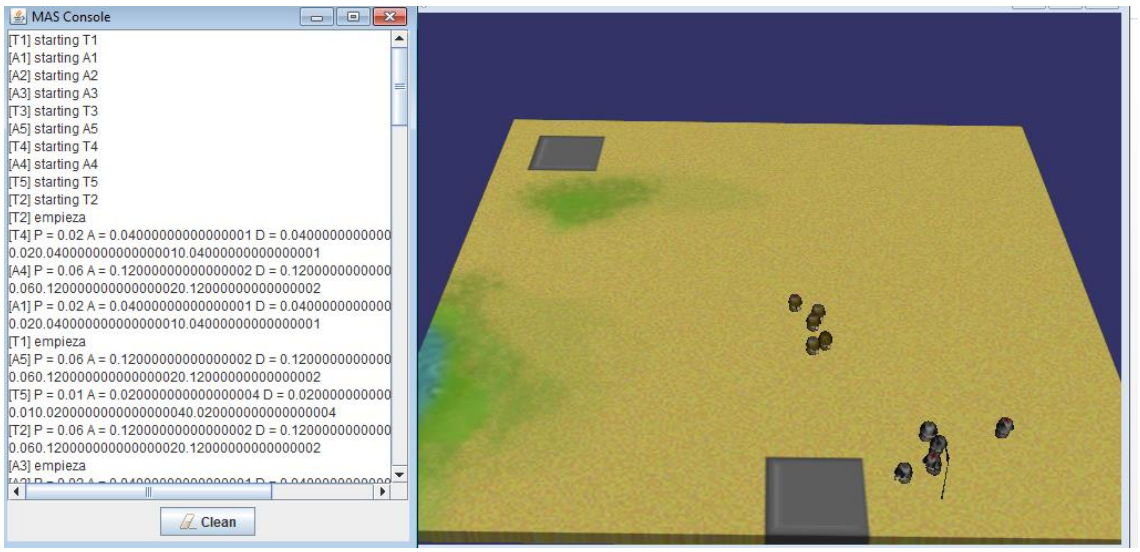

Ilustración 29: Empieza la ejecución de la prueba 1.

En la ilustración 29 se puede observar como los agentes del equipo *Allied* se generan en su base, que está situada en el centro del mapa, al igual que los del equipo *Axis* cuya base se encuentra en la esquina inferior derecha, justo donde se sitúa la bandera.

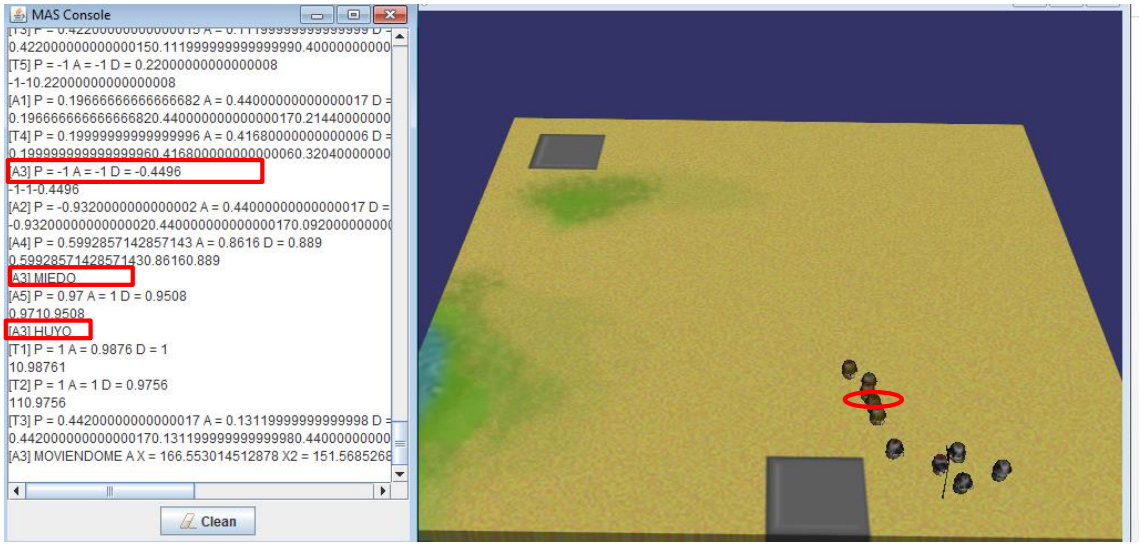

Ilustración 30: El agente A3 (Soldier Allied cobarde) empieza a huir.

En la ilustración 30 se puede observar como el agente 'A3' al tener sus componentes 'P' y 'A' del PAD con valor igual a -1, entra en estado de miedo, además de huir.

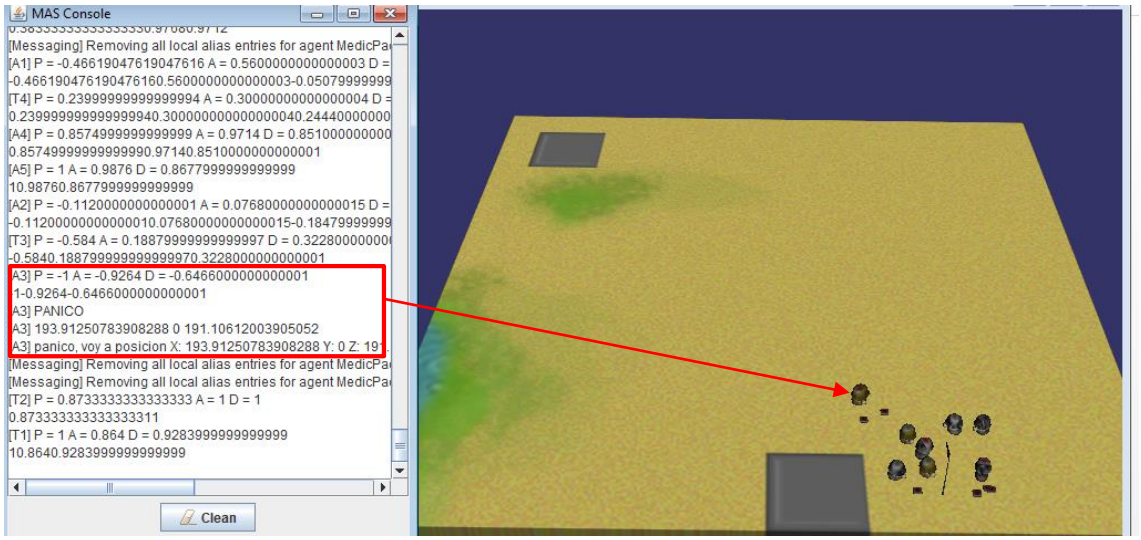

Ilustración 31: El agente A3 entra en pánico.

En la ilustración 31 el agente 'A3' entra en pánico dado que sus tres componentes del PAD están por debajo del umbral (-0.5) y por ello se desplaza a una posición aleatoria.

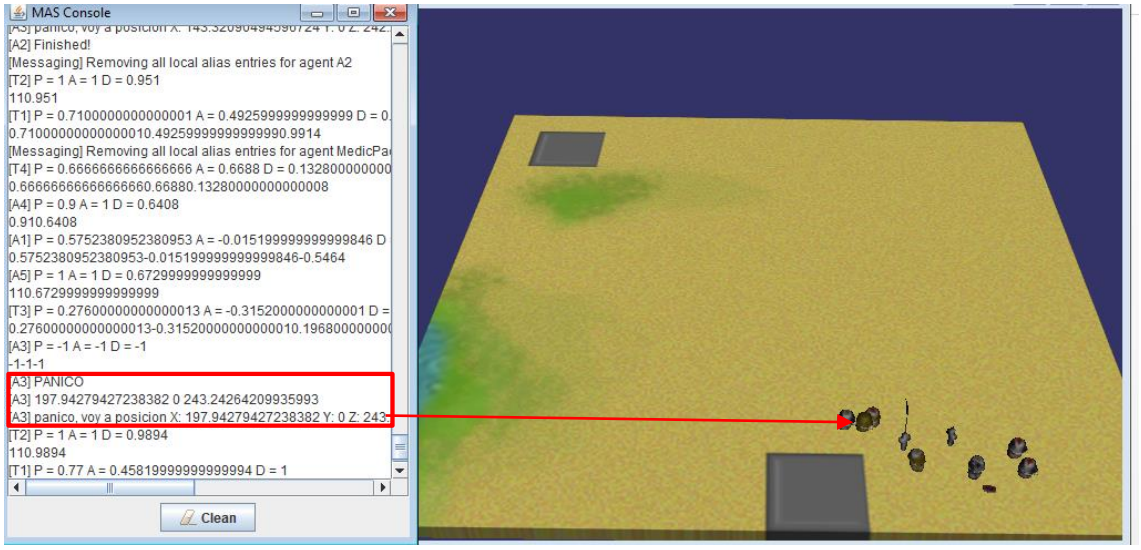

Ilustración 32: El agente A3 corre aleatoriamente.

En la ilustración 32 el agente 'A3' sigue teniendo pánico y por ello sigue corriendo de forma aleatoria.

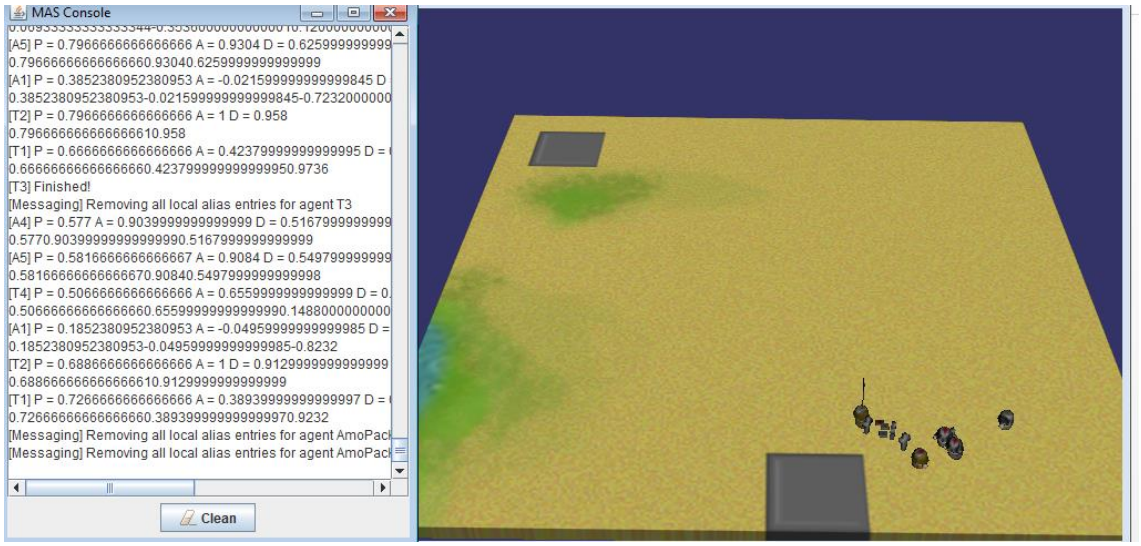

Ilustración 33: Un agente Allied consigue coger la bandera.

En la ilustración 33 se observa como el agente 'A1' ha cogido la bandera y se la lleva hacia su base.

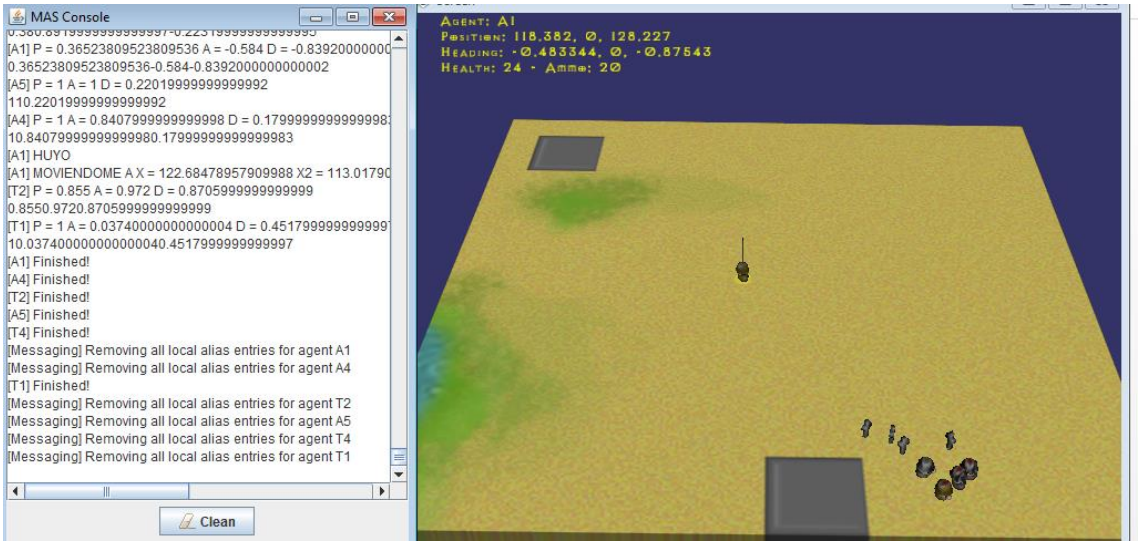

Ilustración 34: Finaliza la ejecución, los aliados has ganado.

En esta prueba han muerto tres agentes de los aliados y dos agentes del eje. Además el agente cobarde del equipo *Allied* ha entrado en pánico, sin embargo eso no ha impedido que los aliados ganaran. En la ilustración 30 se puede ver que el agente 'A3' empieza a huir cuando al menos dos de sus componentes del PAD son inferiores al umbral y en la ilustración siguiente se puede ver que el agente 'A3' tan solo entra en pánico cuando sus tres componentes del PAD son inferiores a -0.5 por lo que actúa conforme a lo establecido.

5.6.2 Segunda Prueba

En esta prueba le he dado una ventaja al equipo del eje dado que cuentan con 8 soldados mientras que el equipo de los aliados cuenta con tan solo 3. El equipo del eje lo forman un agente *medic* con personalidad por defecto, otro *medic* valiente, un *fieldops* defecto y otro cobarde y 4 agentes *soldier*. Entre los *soldier* hay un agente poco cobarde, uno poco valiente, uno valiente y otro cobarde.

Por el otro lado en el equipo del eje hay un agente *soldier* poco cobarde, un agente *medic* poco valiente y un agente *fieldops* con personalidad por defecto.

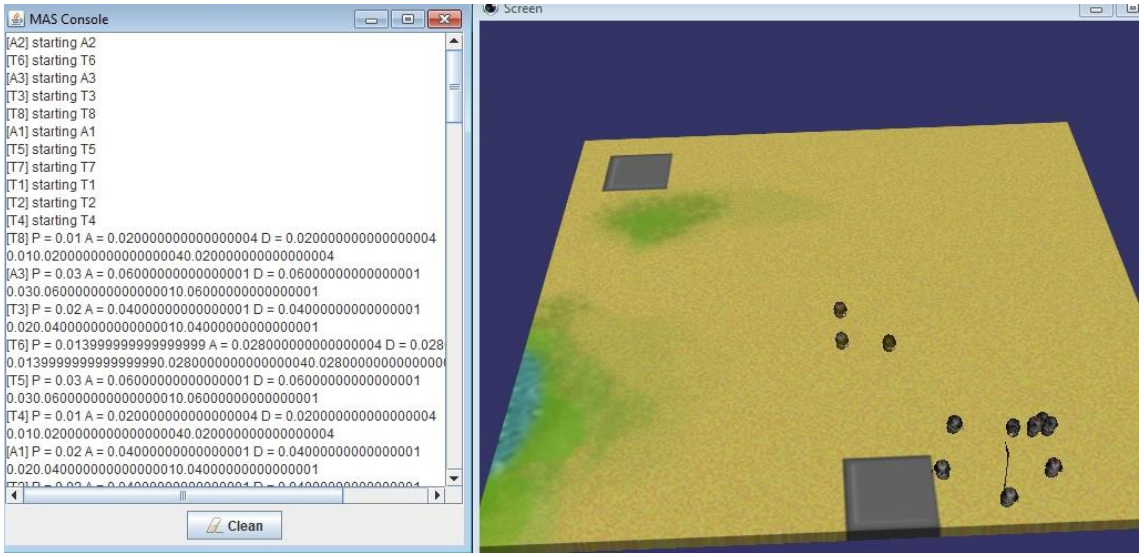

Ilustración 35: Empieza la ejecución de la prueba 2.

En la ilustración 35 se puede observar como los distintos agentes se generan en sus respectivas bases. Hay que tener en cuenta que, según el funcionamiento de JGOMAS, la posición inicial de los agentes se genera aleatoriamente, por lo que en dos ejecuciones distintas los mismos agentes pueden tener posiciones iniciales distintas.

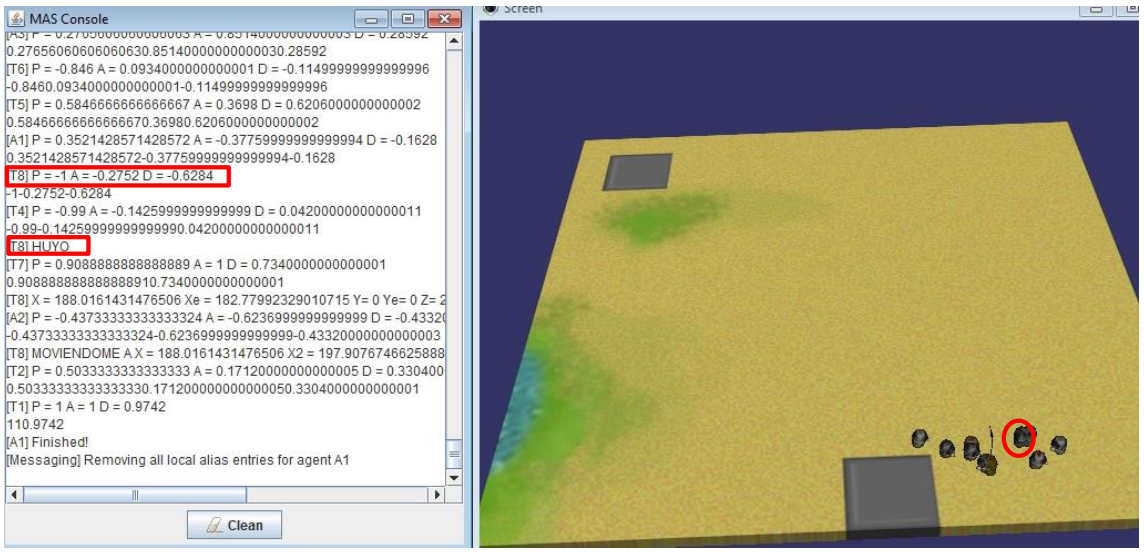

Ilustración 36: El agente T8 (Soldier Axis cobarde) empieza a huir.

En la ilustración 36 se puede ver como el agente 'T8' huye hacia una posición segura al tener sus componentes 'P' y 'D' por debajo del umbral.

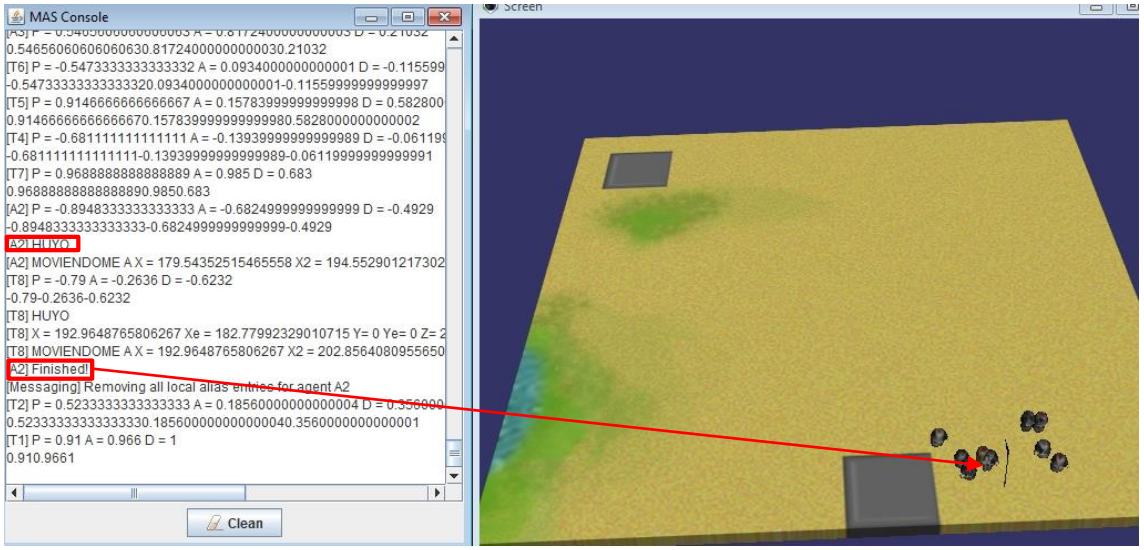

Ilustración 37: El agente A2 (Soldier Allied poco cobarde) muere mientras huye.

En la ilustración 37 se presencia la muerte del agente 'A2' dado que su vida llega al valor cero. La muerte del agente se confirma por el mensaje de '*Finished!*' mostrado.

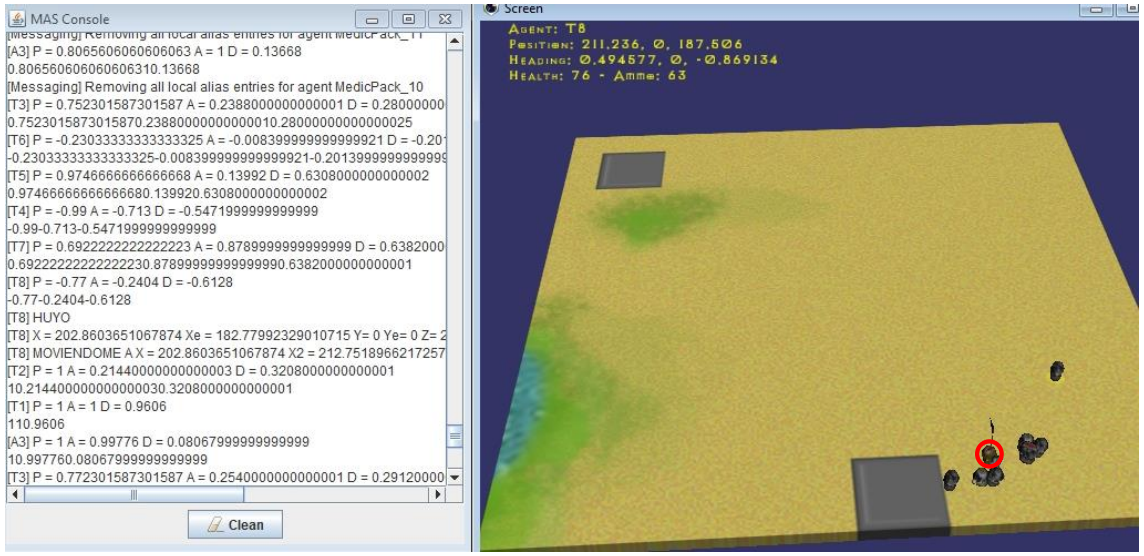

Ilustración 38: El agente A3 coge la bandera.

En la ilustración 38 se observa como el agente 'A3' ha conseguido coger la bandera a pesar de estar rodeado por agentes del eje. Esto puede ser debido a que como la partida se encuentra en un estado avanzado, los agentes enemigos pueden estar escasos de munición y prefieren no dispararle.

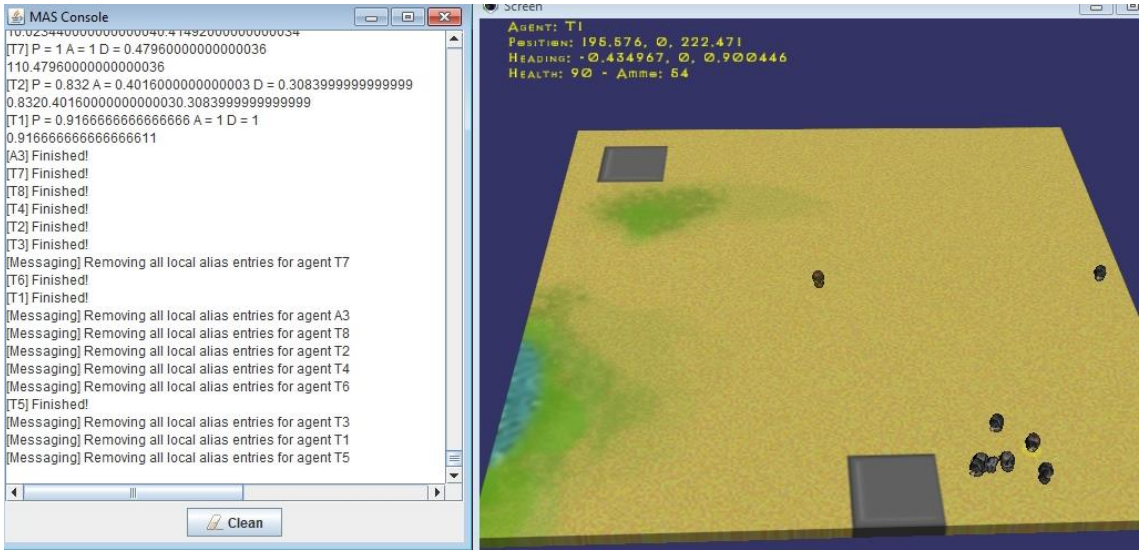

Ilustración 39: Los aliados ganan la partida.

En las ilustraciones 36 y 37 se puede observar que el agente cobarde del equipo del eje empieza a huir antes que el agente poco cobarde del equipo de los aliados, a pesar de que estos se encuentran en clara desventaja. Esto es debido a que los agentes cobardes se asustan mucho más rápido que los otros agentes y por eso muchas veces los agentes poco cobardes o por defecto mueren antes de llegar a sentir pánico.

También se puede ver que a pesar de estar en desventaja, los aliados han ganado y esto puede deberse a que a veces los agentes prefieren no disparar ni perseguir al enemigo dependiendo de su nivel del PAD.

5.6.3 Tercera prueba

En esta prueba he montado el escenario en el que los agentes del eje están en desventaja con respecto a los del equipo de los aliados. En el equipo *Allied* hay un agente *fieldops* poco cobarde, otro cobarde, un agente *soldier* cobarde, otro poco cobarde, otro valiente y otro poco valiente; y un agente *medic* poco valiente y otro valiente.

En el equipo del eje hay un agente *medic* con personalidad por defecto, un agente *soldier* medio valiente y un agente *fieldops* medio cobarde.

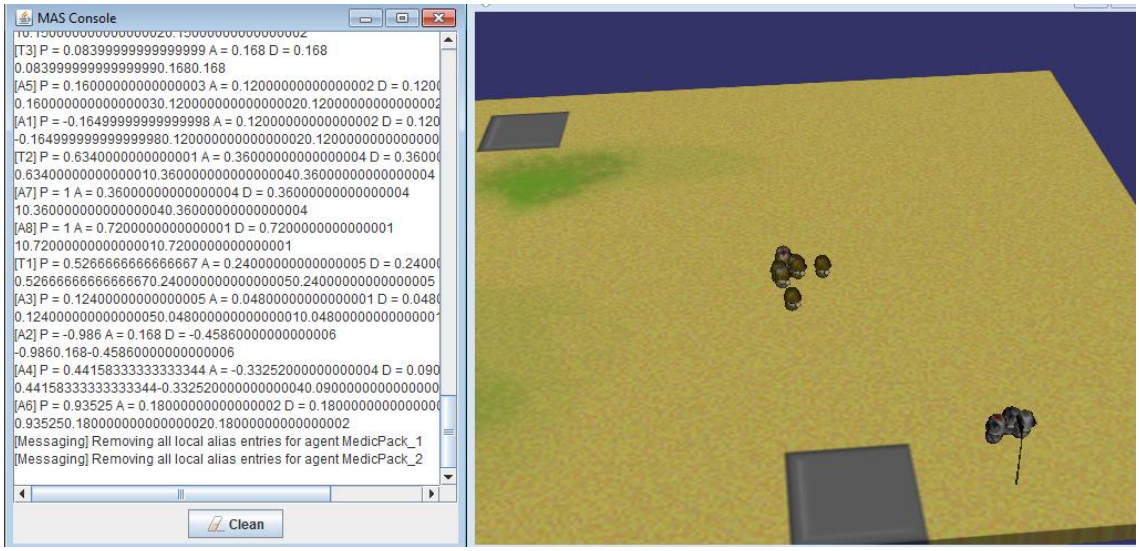

Ilustración 40: Comienza la ejecución de la prueba 3.

En la ilustración 40 se puede observar como los agentes del eje se han generado en una ubicación parecida (los tres agentes se encuentran muy juntos). Por otra parte, los agentes del equipo de los aliados se han generado en forma de 'L', teniendo a los agentes *soldier* en los extremos.

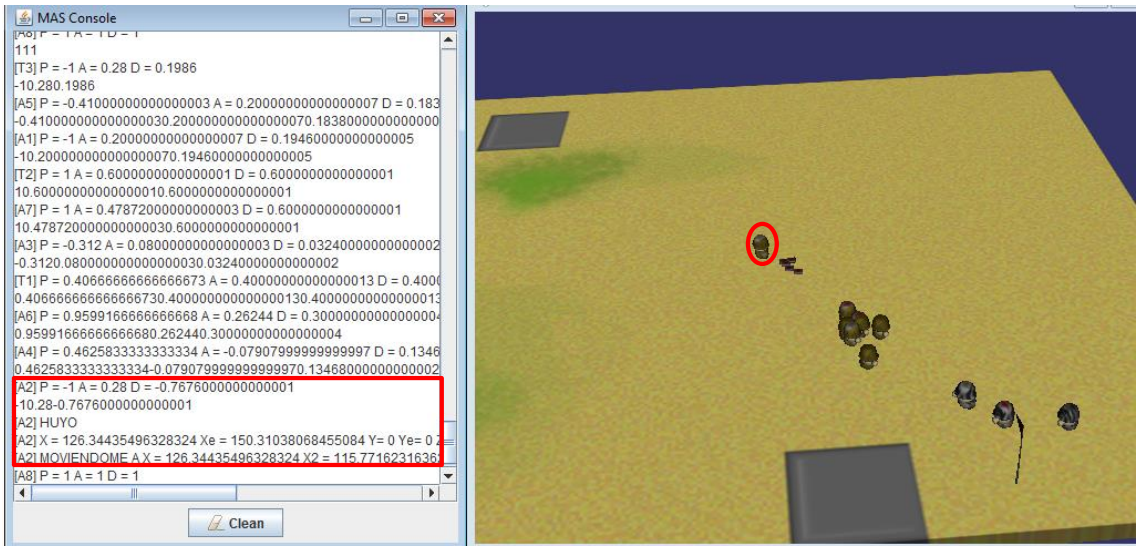

Ilustración 41: El agente A2 (Fieldops Allied poco cobarde) huye.

En la ilustración 41 se puede apreciar como las componentes 'P' y 'D' del agente 'A2' están por debajo del umbral y por ello huye aunque acabe de empezar la partida.

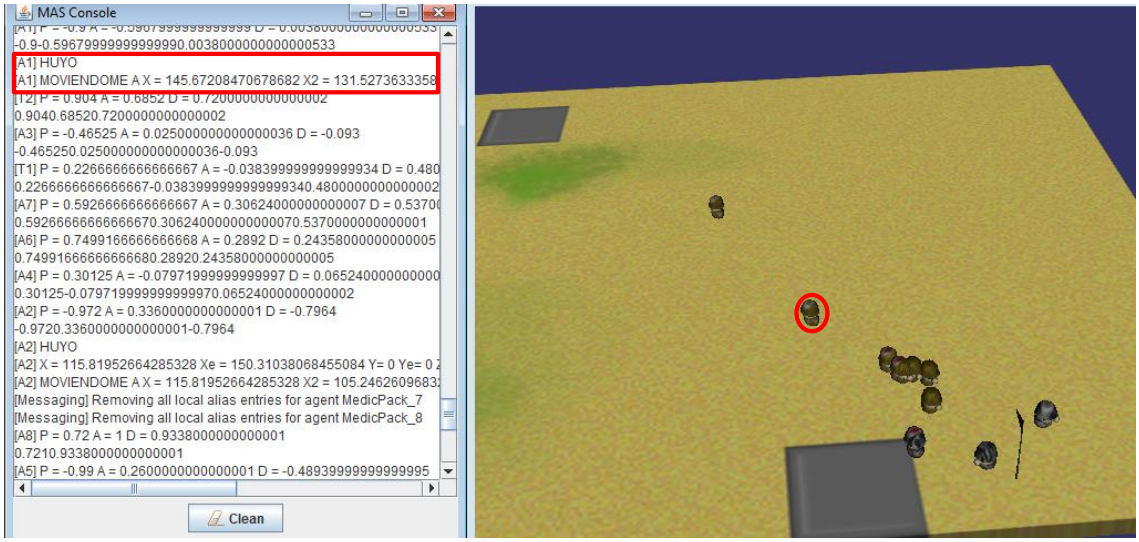

Ilustración 42: El agente A1 (Fieldops Allied cobarde) también huye.

En la ilustración 42 se puede apreciar como el agente 'A1' huye del mismo modo que el agente 'A2' cuando su valor PAD cae por debajo del umbral, a pesar de que no se pueda apreciar correctamente en la consola.

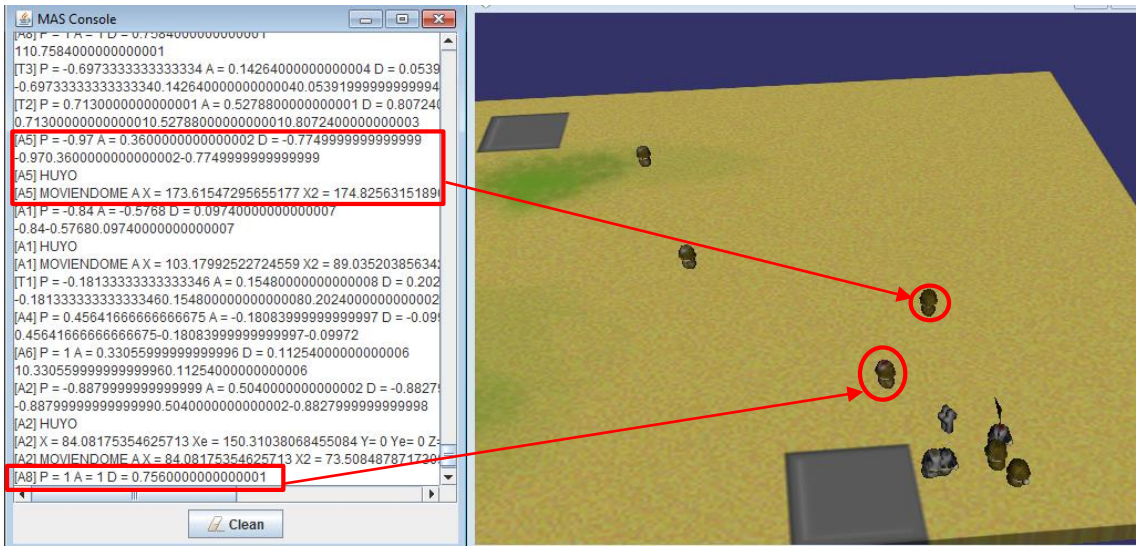

Ilustración 43: El agente A5 huye y el agente A8 va a ayudar a otro agente.

En la ilustración 43 se puede observar que, aunque parezca que el agente 'A8' huya de la misma forma que huye el agente 'A5', en realidad va a ayudar a un compañero, pues sus 3 componentes del PAD tienen valores positivos.

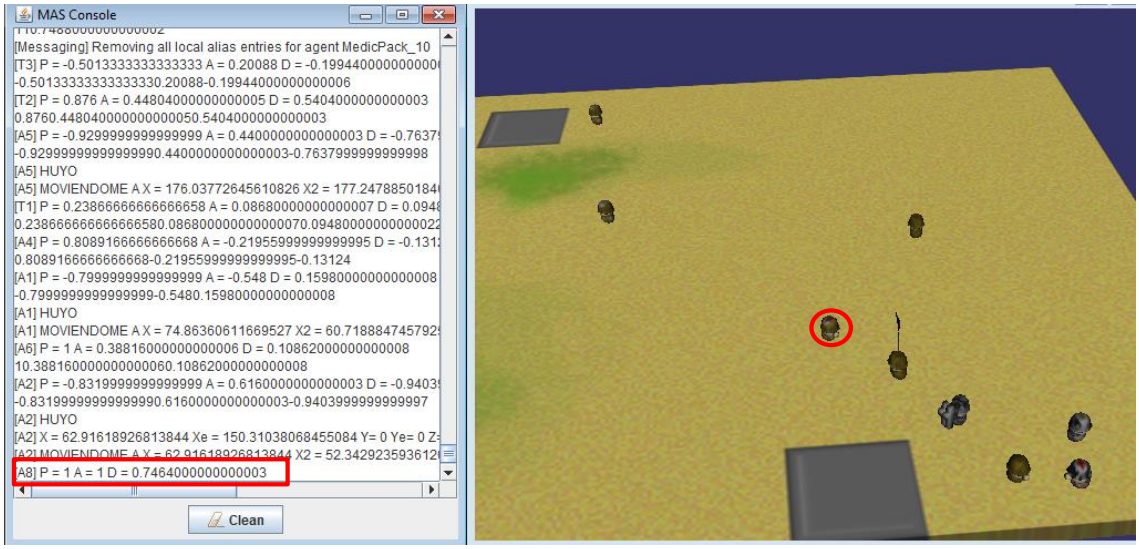

Ilustración 44: El agente A8 deja de ayudar.

En la ilustración 44 se puede apreciar como el agente 'A8' deja de intentar ayudar a sus compañeros por la distancia a la que se encuentran.

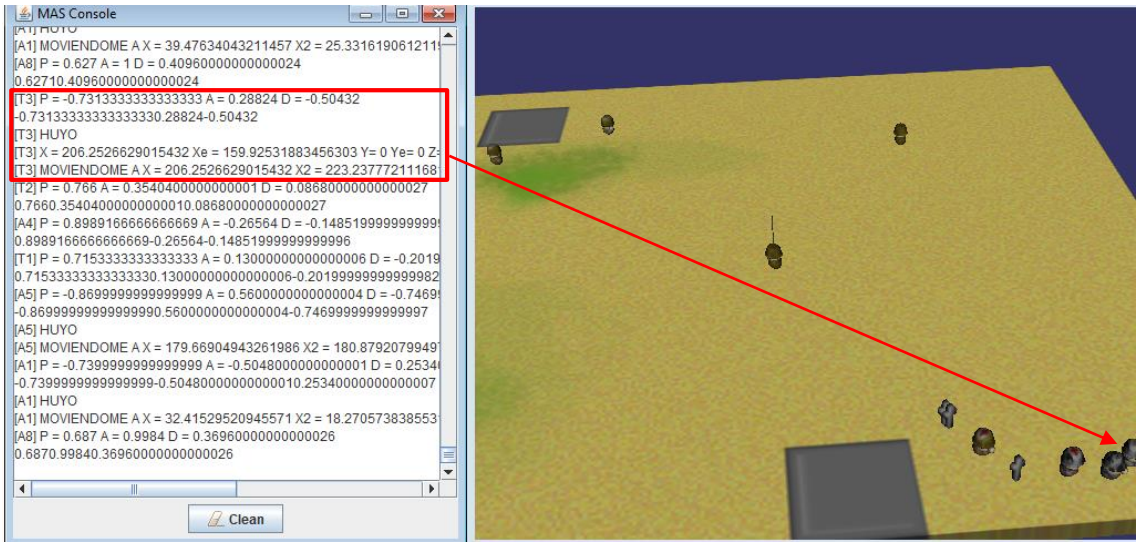

Ilustración 45: El agente T3 (Fieldops Axis medio cobarde) huye.

En la ilustración 45 se puede observar como finalmente el agente 'T3' huye al ver al agente 'A8' y tener sus componentes 'P' y 'D' por debajo del umbral (-0.5).

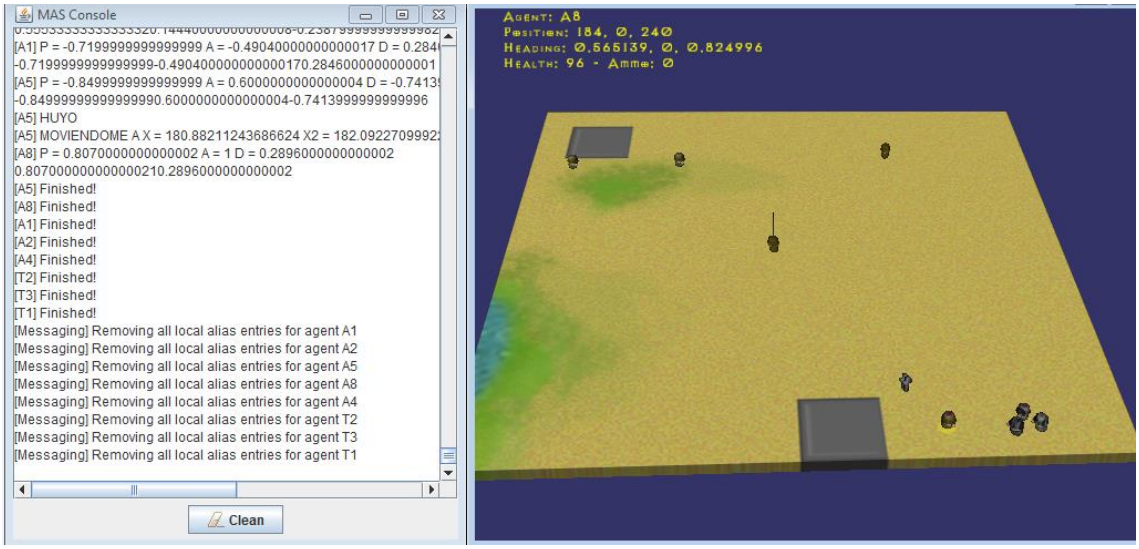

Ilustración 46: Los aliados ganan la partida.

En esta prueba se han dado varios casos curiosos. Primero, el agente medio cobarde ha huido antes que el agente cobarde (ilustraciones 41 y 42). Segundo, el agente *medic* valiente de los aliados ha ido a ayudar a un compañero que huía y al ver que no podía alcanzarlo, ha vuelto a ir a la base enemiga (ilustraciones 43 y 44). Además los agentes del eje no han huido hasta el final a pesar de estar en clara desventaja y esto se debe a que la mayoría de los agentes del equipo de los aliados han huido antes de llegar a su base.

5.7 Evaluación

Si nos fijamos en las pruebas realizadas podemos comprobar que los agentes se comportan del modo en el que había previsto. Hay que tener en cuenta que los agentes sólo pueden saber si tienen cerca un compañero si lo están viendo en ese momento (los agentes ven en un cono delante de ellos). Por esto mismo pueden tener cambios bruscos en la componente 'P' del PAD, aun si son agentes cobardes o valientes.

En las pruebas se observa que conforme los agentes gastan balas cada vez disparan menos. Los agentes que antes se quedan si balas son los agentes valientes, si observamos la ilustración 46 vemos que al final de la partida el agente 'A8' tenía 96 puntos de vida y ninguna bala a pesar de estar en medio de la base del eje. Esto se debe a que como su valor 'D' es muy alto no tiene la necesidad de pedir munición (aunque la necesite realmente). Por otra parte, su vida es muy alta por que los agentes del eje prefieren guardarse sus balas.

Siguiendo esta relación, a los agentes de tipo *medic y fieldops* les afectan de una forma especial las componentes del PAD pues en las pruebas realizadas se puede observar que conforme los agentes van teniendo miedo cada vez ayudan menos a sus compañeros. Por eso los agentes que suelen ayudar casi siempre son los agentes valientes como se puede ver en la ilustración 43.

En otro orden de cosas, tal como se ha visto en las pruebas los agentes cobardes son los que suelen huir antes del campo de batalla (que era el objetivo buscado), mientras que los agentes valientes tienden a ir siempre a por el enemigo aunque sus puntos de vida y munición sean escasos.

Aunque es difícil que ocurra, en la primera prueba un agente *Allied* ha entrado en estado de pánico. Con esto se demuestra que los agentes pueden alcanzar también este estado además de los otros, aunque con menor frecuencia por las condiciones que requiere. Como los agentes suelen huir antes que entrar en el estado de pánico es difícil que alcancen este estado dado que al darse la vuelta pierden por un lado de vista a los enemigos, con lo que no disparan ni pierden munición, y por otro lado los enemigos no pueden dispararle por que se aleja de estos. Es por esto que para alcanzar el estado de pánico debe de encontrarse con muchos enemigos de cara y perder una gran cantidad de vida y munición en un espacio de tiempo muy corto.

6 Conclusión

Si recapitulamos, en este trabajo se han definido una serie de personalidades basadas en estados emocionales para posteriormente incorporarlas a los agentes de JGOMAS. Estas personalidades permiten distinguir a varios tipos de agentes que aunque en un principio pueden resultar muy parecidos, cuando se observa su comportamiento en un entorno real se comprueba lo diferenciados que están.

En las pruebas se puede comprobar que al variar solo el valor de modificación del PAD (factor de crecimiento positivo y negativo) se pueden obtener combinaciones interesantes pues el cambiar este factor no afecta únicamente a la velocidad a la que cambian de estado emocional, sino que puede provocar que por ejemplo un agente amistoso decida no ayudar a un compañero o incluso que no quiera disparar al enemigo (para ahorrar balas). Esto da infinidad de posibilidades para las partidas que además aumenta la inmersividad del usuario pues le es más difícil predecir lo que harán los agentes.

Si a esto le añadiésemos aparte la posibilidad de introducir personalidades aleatorias, es decir, que en cada ejecución de JGOMAS la personalidad que incorpora cada agente fuese puesta automáticamente por el juego, la predicción por parte del usuario sería nula. Al no saber qué tipo de personalidad tiene cada agente el juego se vuelve más realista dado que en el mundo real no se puede saber cómo va a actuar una persona desconocida ante una determinada situación (a menos que se conociese la personalidad de esa persona con anterioridad).

Por otra parte, se ha intentado simplificar lo máximo posible el cálculo de las componentes del PAD de los agentes, haciendo que siempre se calcule de la misma forma (los eventos negativos disminuyen el nivel del PAD y los positivos lo incrementan) y que los agentes siempre actúen de la misma forma ante los mismos estados emocionales. En ampliaciones futuras, se podrían crear agentes con otros tipos de personalidad como por ejemplo, un tipo de agente con una personalidad psicópata, el cual solo se centra en luchar y nunca ayuda a sus compañeros, además de volverse más agresivo conforme tenga ansiedad. Asimismo, se podría crear un agente pacífico que solo dispare cuando tenga ansiedad y su prioridad siempre sea ayudar a sus compañeros o coger la bandera.

Finalmente, tras las evaluaciones realizadas podemos concluir que se han alcanzado los objetivos definidos en la introducción, pues se ha logrado modelar los 5 tipos de personalidades e implementarlas en los agentes de JGOMAS con éxito. Las pruebas realizadas demuestran que los agentes se comportan de la forma propuesta, si los agentes son cobardes perciben el miedo con mayor rapidez que si fueran valientes. Por otra parte, se ha comprobado que cuando se

cumplen las condiciones indicadas, los agentes huyen del enemigo y pueden incluso entrar en estado de pánico, corriendo hacia posiciones aleatorias. De la misma forma, cuando los agentes sufren estrés disparan con menos frecuencia para ahorrar munición y los agentes *medics* y *fieldops* dejan de ayudar a sus compañeros.

Como trabajo futuro se podría ahondar en el tema de la dinámica emocional, donde entra la característica de la empatía que consiste en la capacidad de un agente de percibir el estado emocional en el que se encuentra otro agente. Así pues, los agentes podrían actuar guiándose por lo que sienten sus compañeros, es decir, por cómo se encuentren las componentes del PAD de sus compañeros, en lugar de únicamente por cómo se sienten ellos mismos. Junto con la dinámica emocional, se podría implementar además un sistema de contagio emocional entre los agentes. En el contagio emocional el agente no solo percibe el estado emocional de sus compañeros sino que además se contagia de ese estado, es decir, que los valores de las componentes del PAD del agente compañero influyen de alguna forma en las componentes del PAD del propio agente. Por ejemplo, si un agente determinado se acerca a su compañero y este tiene miedo, el agente en cuestión comenzará a sentir miedo y al cabo de un tiempo se encontrará en un estado parecido al de su compañero.

7 Bibliografía

1. Wikipedia. [En línea] 6 de 7 de 2015. https://es.wikipedia.org/wiki/Algoritmo_de_la_colonia_de_hormigas.

2. *Agentes Inteligentes: el siguiente paso en la inteligencia artificial.* V. Julián, V. Botti. Especial 25 aniversario, Barcelona : ATI, 2000, NOVATICA.

3. biblioteca itam. [En línea] 1987. http://biblioteca.itam.mx/estudios/estudio/estudio10/sec_16.html.

4. linamce. monografias. [En línea] 18 de Octubre de 2004. http://www.monografias.com/trabajos16/la-inteligencia-artificial/la-inteligencia-artificial.shtml.

5. Franklin, S. y Graesser, A. Is It an agent, or just a program?: A taxonomy for autonomous agents. *Intelligent Agents III Agent Theories, Architectures, and Languages*. Heidelberger : Springer Berlin Heidelberg, 1996.

6. Nwana, Hyacinth S. Software agents: an overview. AA&T, BT Laboratories, Ipswich, United Kingdom : Cambridge University Press, 1996.

7. Wikipedia. *Belief–desire–intention software model.* [En línea] 4 de 7 de 2015. [Citado el: 30 de 7 de 2015.] https://en.wikipedia.org/wiki/Belief%E2%80%93desire%E2%80%93intention_software_model.

8. Alfonso Espinosa, B., Vivancos Rubio, E. y Botti Navarro, VJ. *Extending a BDI agents' architecture with open emotional components*. Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València. 2014.

9. Guerra-Hernández, A., Fallah-Seghrouchni, A. El y Soldano, H. *Learning in BDI Multi-agent Systems.* Heidelberger : Springer Berlin Heidelberg, 2005. ISBN: 978-3-540-24010-5.

10. Castelfranchi, Cristiano. *Affective Appraisal versus Cognitive Evaluation in Social Emotions and Interactions.* Heidelberger : Springer Berlin Heidelberg, 2000. ISBN: 978-3-540-41520-6.

11. Jason. [En línea] [Citado el: 31 de Julio de 2015.] http://jason.sourceforge.net/wp/description/.

12. Rafael H. Bordini, Jomi F. Hübner. Jason Documents. [En línea] 0.9.5, Febrero de 2007. [Citado el: 1 de Septiembre de 2015.] http://jason.sourceforge.net/Jason.pdf.

13. GTI IA. *Grupo de Tecnológia Informática e Inteligencia Artificial.* [En línea] [Citado el: 18 de 8 de 2015.] http://gti-ia.upv.es/sma/tools/jgomas/archivos/documentation/Jason%20JGOMAS%20-%20Manual.pdf.

14. Dietrich, D., Fodor, G., Zucker, G., Bruckner, D. (Eds.), [ed.]. *Simulating the Mind: A Technical Neuropsychoanalytical Approach.* s.l. : Springer-Verlag Wien, 2009.

15. C. Mourlas, N. Tsianos, P. Germanakos, [ed.]. *Cognitive and Emotional Processes in Web-Based Education: Integrating Human Factors and Personalization.* s.l. : IGI Global, 2009.

16. Mehrabian, Albert. *Analysis of Affiliation-Related Traits in Terms of the PAD Temperament Model, The Journal of Psychology: Interdisciplinary and Applied*. 1997. págs. 101-117.

17. Gebhard, Patrick. *ALMA – A Layered Model of Affect*. German Research Center for Artificial Intelligence (DFKI). Saarbrücken : s.n., 2005.

18. Wikipedia. *PAD emotional state model.* [En línea] 28 de Junio de 2015. https://en.wikipedia.org/wiki/PAD_emotional_state_model.