UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Implementación del juego jGomas en Spade 3

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Navarro Osma, Raúl

Tutor/a: Julian Inglada, Vicente Javier

Cotutor/a: Carrascosa Casamayor, Carlos

Director/a Experimental: PALANCA CAMARA, JAVIER

CURSO ACADÉMICO: 2022/2023

Resumen

En este trabajo se analiza la versión actual de pyGOMAS, un juego basado en agentes inteligentes utilizando SPADE y desarrollado en Python. Se propone la implementación de una nueva versión que proporcione a los estudiantes ciertas herramientas que faciliten la configuración y ejecución de todas las partidas, además de interactuar con ellas a tiempo real, mejorando de esta manera la eficiencia, efectividad y la satisfacción del usuario. Para lograr esto, se añade una interfaz gráfica como apoyo a los usuarios, proporcionando una lista de funcionalidades que logren satisfacer todos los requisitos planteados. La intención es proporcionar a los estudiantes una herramienta útil para mejorar el trabajo práctico en las asignaturas relacionadas directamente con las IA, como, por ejemplo, Agentes Inteligentes (AIN)

Palabras clave: PYGOMAS, SPADE, Sistema multiagente, Agentes Inteligentes.

Abstract

In this task, the current version of pyGOMAS, a game based on intelligent agents using SPADE and developed in Python, is analyzed. We propose the implementation of a new version that provides students with certain tools to facilitate the configuration and execution of all games, as well as interact with them in real-time, thereby improving efficiency, effectiveness, and user satisfaction. To achieve this, a graphical interface is added to support users, providing a list of functionalities that meet all the requirements set forth. The intention is to provide students with a useful tool to enhance practical work in subjects directly related to AI, such as Intelligent Agents (IA).

Keywords: PYGOMAS, SPADE, Multi-Agent system., Intelligent Agents.

Tabla de contenidos

Índice de figuras

1. Introducción

Es un hecho que la Inteligencia Artificial (IA) es un campo de la informática que está en continuo crecimiento y levanta mucho interés. Esta disciplina busca desarrollar sistemas y programas capaces de realizar tareas que normalmente requerirían de la inteligencia humana. A medida que avanzamos en la era digital, la IA se ha convertido en un componente principal de numerosas aplicaciones y servicios que utilizamos en nuestra vida diaria. [1]

Desde las clásicas IA basada en reglas hasta el aprendizaje automático (Machine Learning), cada tipo de IA tiene sus propias características y capacidades. Explorar estos tipos no solo proporciona una comprensión más profunda de las capacidades actuales de la tecnología, sino que también brinda conocimientos para potenciar las posibilidades que la IA podrían ofrecer en un futuro.

1.1 Motivación

La principal motivación en este proyecto surge de la necesidad de poder brindar a los estudiantes de Ingeniería Informática que cursen asignaturas directamente relacionadas con los agentes inteligentes, una herramienta efectiva para el manejo de dichos agentes a través de pyGOMAS, un juego basado en el clásico modo de capturar la bandera donde participan distintos agentes inteligentes.

PyGOMAS es la base de las prácticas de esta asignatura, donde se trabaja con una gran variedad de agentes inteligentes con distintas funciones. Este proyecto busca incrementar la eficiencia de los estudiantes a la hora de realizar experimentos prácticos a través de una interfaz de usuario que permita configurar, ejecutar, crear y visualizar partidas en tiempo real de pyGOMAS. Con esta herramienta se pretende capacitar a los estudiantes para que se conviertan en creadores y usuarios competentes de agentes inteligentes.

Por otra parte, la motivación de este proyecto también viene dada por el interés propio de explorar y aprender tecnologías y conceptos poco utilizados en el ámbito personal, ya que, supone un desafío apasionante adentrarse en terrenos menos explorados, como los agentes inteligentes, código asíncrono o lenguajes poco utilizados como en este caso Python. Esto puede brindar un gran aprendizaje en términos de resolución de problemas de manera innovadora y la adquisición de nuevas habilidades.

1.2 Objetivos

El objetivo del proyecto es crear una nueva herramienta que funcione en paralelo con pyGOMAS, y que proporcione a los estudiantes una serie de comodidades para poder realizar pruebas y exprimir mejor todas las características del juego basado en agentes inteligentes.

Se han planteado una serie de objetivos más concretos para que se pueda cumplir el objetivo global:

- Analizar las necesidades de los estudiantes para una mejorar el manejo de pyGOMAS.

- Implementar una interfaz que facilite todas las tareas de ejecución configuración y creación de una partida de pyGOMAS.
- Realizar la implementación de todas las funcionalidades propuestas.
- Identificar las funcionalidades que no han sido implementadas que puedan ser abordadas en un trabajo futuro.

1.3 Estructura de la memoria

En este apartado se presentan los distintos capítulos que se encontrarán a lo largo de la memoria y una breve descripción de lo que podemos encontrar en cada uno de ellos:

Capítulo 1: Introducción. Se ha presentado una breve introducción del proyecto, la principal motivación para realizarlo y los objetivos que se pretenden alcanzar.

Capítulo 2: Estado del arte. Se realizará una descripción detallada de los principales componentes que han tenido un gran impacto durante el desarrollo del proyecto.

Capítulo 3: Análisis y diseño. En este capítulo se realiza un análisis de requisitos, se comenta la arquitectura del proyecto realizado y las tecnologías utilizadas para llevarlo a cabo.

Capítulo 4: Desarrollo de la solución propuesta (Primera versión). Se mostrará todo el desarrollo de la primera versión del proyecto, así como las principales funcionalidades implementadas y como se ha realizado dicho desarrollo.

Capítulo 5: Desarrollo de la solución propuesta (Segunda versión). Se especificará el motivo de la existencia de una segunda versión del proyecto, los problemas que han surgido y como se ha reconfigurado todo el proyecto. Se seguirán especificando nuevas funcionalidades añadidas y todo el proceso para poder implementarlas.

Capítulo 6: Pruebas. A lo largo de este capítulo se realizarán y se documentarán dos ejecuciones completas del proyecto que abordan todas las funcionalidades implementadas además de otros aspectos más técnicos.

Capítulo 7: Conclusiones. Se realizará una conclusión del proyecto y su resultado, se comentarán oportunidades de mejora para futuras implementaciones y la relación del trabajo con los estudios cursados.

Capítulo 8: Bibliografía. En este capítulo estarán incluidas todas las referencias bibliográficas utilizadas a lo largo de la memoria.

2. Estado del arte

2.1 Sistemas multiagente

Los sistemas multiagente constituyen un área de estudio de gran relevancia en el campo de la inteligencia artificial. Estos sistemas se centran en el análisis y desarrollo de entidades autónomas conocidas como agentes, los cuales interactúan de manera colaborativa para lograr objetivos compartidos. Estos agentes, pueden tomar decisiones y ejecutar acciones de forma independiente, otorgándoles una gran autonomía en la toma de decisiones. [2] Algunas de las características clave de los sistemas multiagente son:

- Autonomía: Los agentes son capaces de tomar decisiones de manera independiente basadas en su propio conocimiento y objetivos.
- Interacción: Los agentes pueden interactuar entre sí para lograr objetivos comunes.
- Cooperación: Los agentes pueden cooperar y compartir información para alcanzar resultados superiores a los que podrían lograr de forma individual.
- Competencia: Los agentes pueden competir entre sí por recursos o para lograr sus propios objetivos.
- Comunicación: Los agentes pueden comunicarse entre sí, intercambiando información y coordinando acciones.
- Adaptabilidad: Los sistemas multiagente son flexibles y pueden adaptarse a cambios en el entorno o en los objetivos ajustando sus estrategias y comportamientos.

Estas características permiten a los agentes abordar problemas de manera efectiva y se podrán encontrar muy frecuentemente a lo largo del proyecto. [3]

Existen varias plataformas populares que se utilizan para el desarrollo y la implementación de sistemas multiagente, algunas de las más conocidas que podemos encontrar son:

- JADE [4] (Java Agent Development Framework), desarrollado en Java. Ofrece un conjunto de herramientas y librerías para facilitar tareas como la comunicación y coordinación de los agentes a través de mensajes, la gestión de agentes donde se incluye el arranque y la parada de estos y la implementación de comportamientos, los cuales definen cómo interactúan los agentes dependiendo de algunos aspectos como el entorno y los objetivos.
- NetLogo [5] es un entorno de modelado y simulación basado en agentes. Se caracteriza por ofrecer una interfaz gráfica, así como un lenguaje de programación específico.
- MASON [6] (Multiagent Simulator of Neighborhoods), plataforma desarrollada en Java. Está diseñada para modelar sistemas multiagente además de proporcionar distintas herramientas para la simulación y análisis de los agentes.
- AnyLogic [7] es una plataforma de simulación que permite diferentes paradigmas. Esta plataforma nos permite la creación de modelos multiagente con la posibilidad de combinar agentes individuales y agentes colectivos, también proporciona herramientas de visualización para la evaluación del comportamiento de los agentes.
- SPADE [8] es el sistema multiagente utilizado durante el proyecto y del cual se hablará detalladamente más adelante. Este sistema está implementado en Python, que puede

ejecutarse de manera distribuida en distintas máquinas e incluso con distintos sistemas operativos.

Dentro de los sistemas de agentes podemos encontrar distintos tipos de agentes entre los que se destacan algunos como los siguientes:

- Agentes basados en reglas: los agentes se basan en un conjunto de reglas predefinidas para tomar las decisiones y realizar acciones.
- Agentes basados en objetivos: estos agentes definen un objetivo y toman decisiones en función de cómo se acercan a dicho objetivo, suelen hacer uso de las circunstancias del entorno u otros agentes para adaptar su comportamiento.
- Agentes reactivos: son agentes que toman decisiones de forma inmediata en función de la información que reciben del entorno o de otros agentes.
- Agentes deliberativos: toman decisiones después de un proceso de recopilación, análisis y evaluación de información. Genera diferentes opciones, las razona y delibera para determinar cuál es la más adecuada, tomando una decisión basada en este proceso.
- Agentes cognitivos: tienen la capacidad de aprender basándose en la experiencia, suelen evaluar si una acción realizada es correcta o no para lograr un objetivo y aprenden a modificar sus propias reglas.

Los agentes pueden adaptar características de distintos tipos y esto dependerá de los requisitos y las características específicas del sistema.

También es importante destacar el paralelismo y la distribución que aprovechan los sistemas ya que pueden ejecutarse en diferentes nodos o máquinas, todo esto permite una mayor capacidad de colaboración y coordinación entre los agentes, además de una adaptación en el entorno dinámico en el que se encuentran.

Todas las ventajas mencionadas permiten que los sistemas multiagente sean capaces de resolver problemas de manera muy eficiente, por este motivo podemos encontrar estos sistemas en una gran variedad de campos como la robótica, la logística, el transporte y gestión de tráfico, economía, medicina y muchos otros sectores.

2.2 SPADE

SPADE es una plataforma de sistemas multiagente basada en servidores XMPP que nos proporciona un gran número de ventajas en comparación con otras de las plataformas mencionadas anteriormente. A continuación, profundizaremos en el funcionamiento de esta plataforma y cómo funcionan los agentes.

Los agentes SPADE están compuestos por un mecanismo de conexión, un sistema de mensajes que permite la comunicación de los agentes y un conjunto de comportamientos capaces de enviar y recibir mensajes. Los agentes se conectan al servidor XMPP mediante un (JID) y una contraseña. El sistema de mensajería es muy similar a los conocidos servidores de correo SMTP [9], permitiendo a los agentes comunicarse con cualquier otro agente existente como se muestra en la Figura 1.

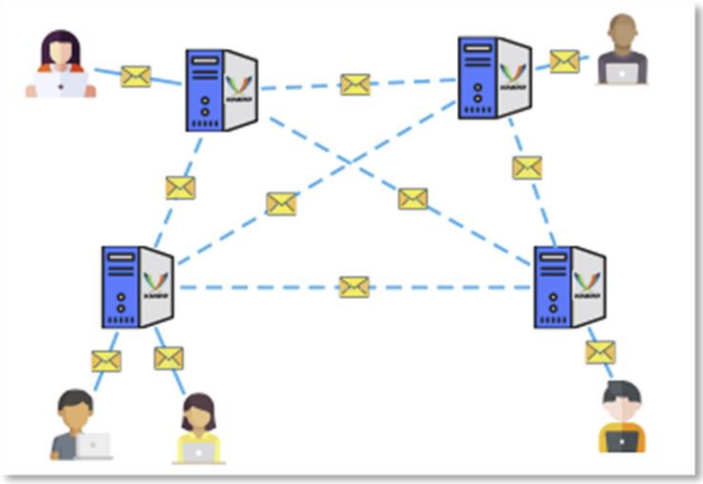


Figura 1. Representación de cómo los servidores XMPP se conectan entre y con los clientes

Los comportamientos forman parte del código ejecutado por los agentes, es el lugar donde se define la acción que un agente debe realizar. Cada agente puede tener varios comportamientos de forma simultánea, lo que nos permite estructurar la lógica de los agentes de una manera muy sencilla. SPADE nos proporciona varios comportamientos:

- One-Shot: es el comportamiento más simple, como indica su nombre es ejecutado una sola vez, muy útiles para realizar tareas ocasionales.
- Periodic: este comportamiento nos permite ejecutar el proceso repetidamente en un periodo programado. Puede ser muy útil para tareas repetitivas
- Time-out: es similar a One-Shot, pero su activación se desencadena en una fecha y hora especificada.
- Finite State Machine: este comportamiento permite a los agentes SPADE construir comportamientos mucho más complejos en nuestro modelo de agente.

A cada uno de estos comportamientos es posible establecer una plantilla, que es el método utilizado por SPADE para enviar y recibir mensajes. Agregar una plantilla a un comportamiento, posibilita al agente enviar mensajes a otros agentes mediante un *dispatcher*, que coloca un mensaje en el flujo de comunicación y lo envía al comportamiento que está esperando dicho mensaje. Los atributos que se pueden establecer en una plantilla son los siguientes:

- To: la cadena JID del destinatario del mensaje
- Sender: la cadena JID del remitente del mensaje
- Body: el contenido del mensaje.
- Thread: el ID del hilo de la conversación
- Metadata: un diccionario (clave, valor) de cadenas para definir metadatos del mensaje. Esto es muy útil, por ejemplo, para incluir atributos FIPA como ontología, performativa, lenguaje, etc.

Una de las principales características de los agentes SPADE es su capacidad para mantener una lista de contactos y recibir notificaciones a tiempo real sobre sus contactos. Esta es una

característica heredada de la tecnología de mensajería instantánea y que, gracias a XMPP, SPADE potencia al máximo para sus agentes. Cada agente cuenta con una función especial para gestionar su presencia, conocida como “presencia”. Este gestor se encarga de administrar todos los aspectos relacionas con la notificación de presencia de un agente. La presencia tiene tres atributos:

- State: el estado de un mensaje de presencia indica si el agente está disponible o no disponible. Esto significa que el agente está conectado a un servidor XMPP o no. Esta información es muy útil para saber, antes de contactar con un agente, si está disponible para recibir un mensaje a tiempo real. El estado de disponibilidad es un atributo booleano.
- Status: se utiliza para establecer un estado textual en la información de presencia, mediante lenguaje natural se explica el estado actual del agente, que se difunde cuando el cliente se conecta y cuando se vuelve a emitir la presencia. Algunos posibles ejemplos podrían ser “inactivo” u “ocupado”.
- Priority: un agente puede tener múltiples conexiones a un servidor XMPP, por este motivo es posible establecer una prioridad de cada una de estas conexiones para determinar su nivel.

Una parte muy importante de los agentes SPADE es que proporcionan una interfaz gráfica a la que se puede acceder a través de la ruta “/spade”. En la siguiente imagen se muestra dicha interfaz gráfica:

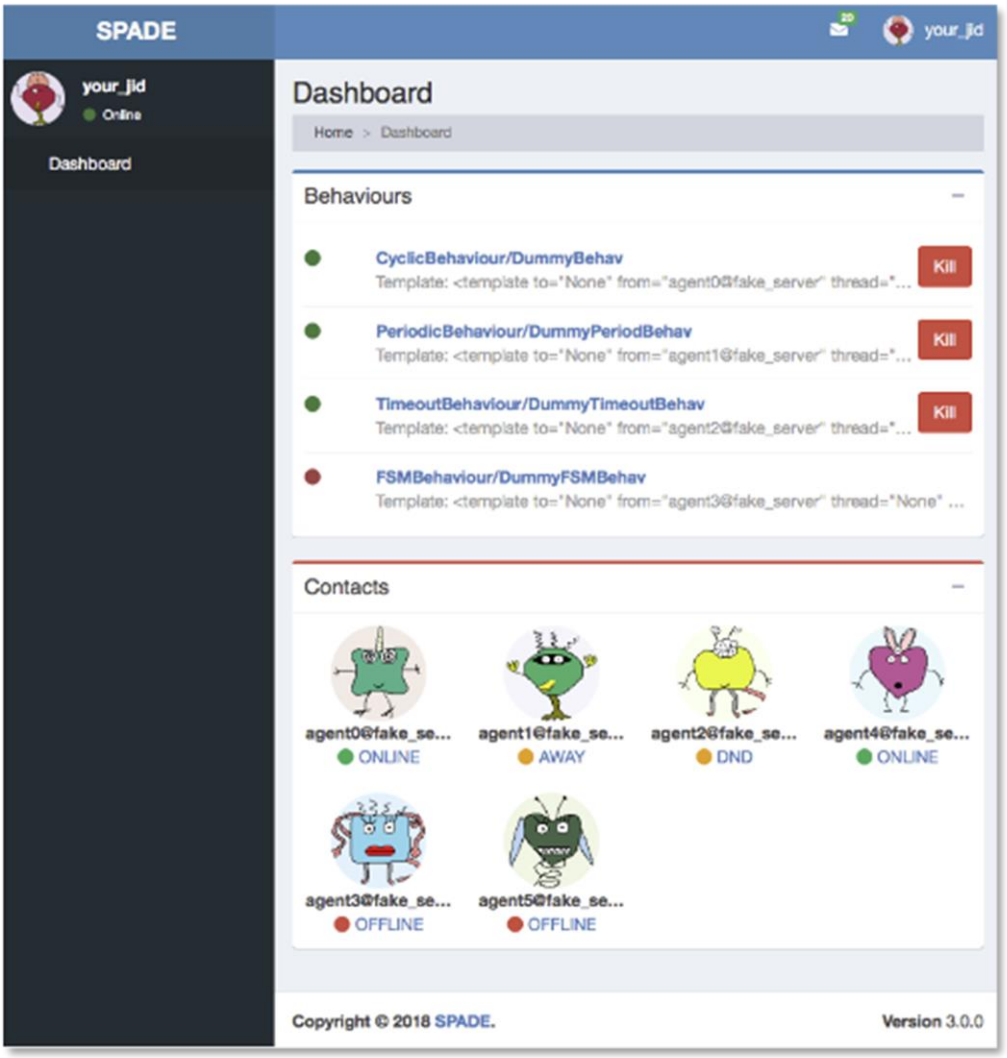


Figura 2. Interfaz gráfica de un agente SPADE

En la imagen, se puede observar la página de índice de un agente, donde se puede verificar su nombre y avatar, una lista de sus comportamientos y una lista de sus contactos. En la barra de menú superior también se pueden ver sus mensajes entrantes, y el menú de perfil del agente, donde se puede detener dicho agente. También es posible clicar en un comportamiento para ver más detalles como el buzón de mensajes, la plantilla y su código de salida entre otros. También es posible ver información adicional de cada contacto del agente.

El módulo web de SPADE también se puede utilizar para crear aplicaciones propias servidas por los propios agentes, se pueden registrar nuevas rutas en el módulo web y, siguiendo el paradigma modelo-vista-controlador (MCV), registrar controladores que calculen los datos necesarios a partir del agente y rendericen una plantilla que se servirá cuando alguien solicite la ruta con la que se registró. Esto último tiene gran importancia en el desarrollo del proyecto.

2.3 PYGOMAS

PyGOMAS es un juego de la modalidad “capturar la bandera” desarrollado en Python y utilizando SPADE como plataforma para sistemas multiagente. También utiliza una arquitectura BDI (Belief-Desire-Intention) y formado por distintos agentes con comportamientos reactivos, deliberativos e híbridos, que permite utilizar las distintas ventajas de ambos tipos de comportamiento. La arquitectura BDI se basa en los componentes de creencias, deseos e intenciones para modelar el razonamiento y el comportamiento de los agentes, permitiéndoles tomar decisiones informadas y adaptarse a un entorno dinámico. [10]

2.3.1 Descripción del juego

Las partidas transcurren en un terreno de juego limitado, donde son participes dos equipos, los Aliados y el Eje. Cada uno de ellos tiene su propia base situada en el un lugar concreto del mapa y están formados por unidades a los que nos referiremos como tropas. En el terreno también se encuentra la bandera, que es el objetivo de ambos equipos.

El equipo Aliado tiene como objetivo común llegar hasta la localización de dicha bandera, capturarla y trasladarla hasta su base. Por otro lado, el equipo Eje tiene como objetivo llegar hasta la bandera y quedarse patrullando cerca de ella para impedir que el equipo rival pueda capturarla y llevar hacia la base rival. La partida termina en caso de que el equipo Aliado consiga llevar la bandera a su base, donde en este caso, ellos saldrán vencedores. Por otra parte, para que el equipo Eje pueda salir victorioso, deben eliminar a todos las tropas del equipo Aliado, o bien deben defender la bandera hasta que el tiempo de duración de la partida finalice.

Para que los equipos puedan conseguir sus objetivos, disponen de armas, munición y unos determinados puntos de salud. Podrán disparar a las otros tropas, restándoles puntos de salud. En caso de que alguna unidad se quede con 0 puntos de salud, quedará eliminada de la partida, impidiéndole interactuar en la misma y simulando su “muerte”. Además, cada unidad puede adoptar un rol distinto. Actualmente podemos encontrar tres tipos de roles en cada tropa:

- Soldados: es la unidad más básica y la que mejores capacidades de combate tiene. Dispone de un arma que causa más daño de lo habitual. Además, pueden recibir llamadas de sus compañeros para que pueda ofrecerles apoyo.

- Médicos: esta unidad dispone de unos paquetes de medicina que proporcionan una cantidad de puntos de salud determinada a aquellas tropas que los recojan. Al igual que los soldados, los médicos también pueden recibir llamada de asistencia médica de cualquier compañero de equipo y disponen de un arma y munición.

- Operadores de campo: su función es proporcionar paquetes de munición a otras tropas de la partida, cuando una unidad obtenga uno de estos paquetes, recuperara una cantidad de munición determinada. Al igual que las otras dos unidades mencionadas, los operadores de campo pueden recibir llamadas de apoyo por parte de sus compañeros y también poseen un arma y munición propia.

Las tropas con capacidad de proporcionar paquetes de ayuda ya sean de medicina o de munición, soltarán dichos paquetes en el suelo del terreno de guerra. Los paquetes se quedarán en una posición exacta hasta que alguna unidad pase por encima obteniendo de esta manera los beneficios

(puntos de salud o munición). Cabe destacar que cualquier tropa puede recoger cualquier paquete. Por poner un ejemplo, si un médico del equipo Eje coloca en el suelo un paquete de medicina, un operador de campo del equipo Aliado lo podrá recoger obteniendo puntos de salud, independientemente de que ese paquete haya sido generado por un médico del equipo rival. A continuación, se muestra un ejemplo visual de una partida de pyGomas:

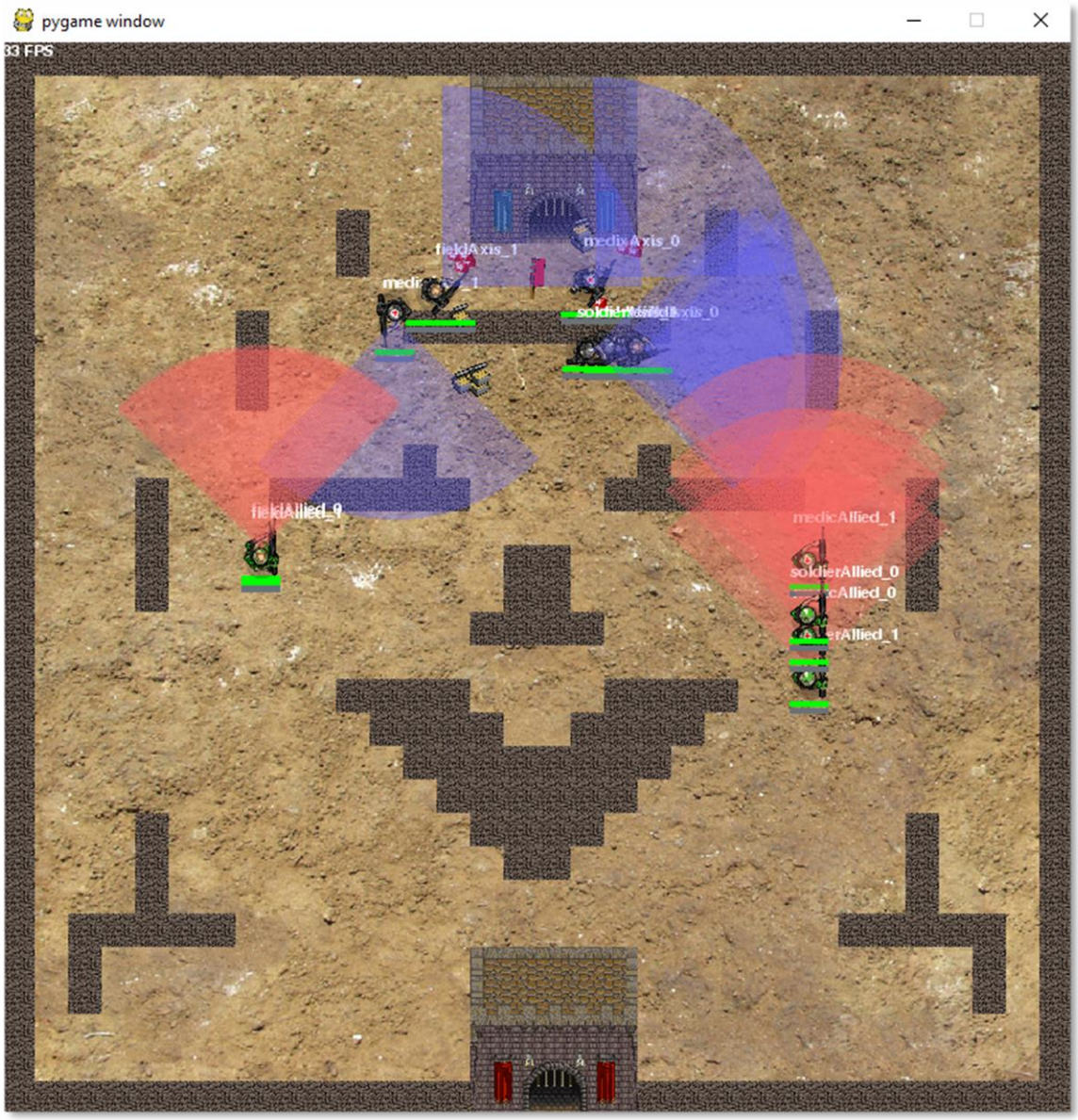


Figura 3. Partida en ejecución de pyGomas.

En la Figura 3 se puede observar cómo las tropas del equipo Aliado se movilizan hacia la posición de la bandera, que está muy cerca de la base del equipo Eje. Las tropas del equipo Eje están patrullando alrededor de la bandera para su defensa. También se pueden observar paquetes de medicina y de munición cerca de la base del equipo Eje (equipo azul).

2.3.2 Agentes de pyGomas

Como se ha mencionado anteriormente, pyGomas está compuesto por distintos agentes que interactúan entre sí, siguiendo la arquitectura deliberativa BDI e incorporando tanto un comportamiento deliberativo como reactivo. A continuación, se van a describir brevemente cuales son estos agentes y cuáles son sus funciones principales.

- **Agente manager**:
  Este agente actúa como gestor durante el transcurso de la partida y la monitoriza. Coordina y sincroniza a los otros agentes y aporta información sobre lo que se encuentra en el campo de visión de las tropas. Podemos encontrar distintas tareas que realiza este agente.

  - Envía un mensaje de inicio de partida a los agentes tropa, indicándoles la posición exacta de la bandera y el terreno de combate sobre el que se desarrolla la partida. También crea un agente de servicios, que se comentará más adelante.

  - Recibe distintos datos relacionados con los agentes tropa de cada equipo, como los puntos de vida, la munición, la posición, etc. El agente manager utiliza estos datos para guiar el transcurso de la partida.

  - Proporciona a los agentes tropa de cada equipo información acerca de los elementos de su campo de visión. Como por ejemplo los paquetes de medicina, paquetes de munición, bandera u otro tipo de agente.

  - Informa a las tropas si han recibido algún paquete de ayuda.

  - Informa a las tropas la cantidad de puntos de salud que poseen. También les informa en caso de que hayan sido heridos.

  - Abre el servidor HTTP para aceptar peticiones de los clientes de visualización. Estos clientes reciben información del manager como la posición de la bandera, la posición de cada una de las tropas y la posición de los paquetes de ayuda.

  - Finaliza la partida informando del equipo ganador. La partida finaliza con la eliminación de todas las tropas del equipo Aliado, la captura de la bandera por parte de los Aliados, o el vencimiento del tiempo de la partida.

- **Agente service:**
  Este agente registra los servicios que ofrecen los agentes tropa. Cuando una tropa es eliminada de la partida, se le informa al agente service para que no pueda ofrecer el servicio que proporcionaba cuando el agente estaba vivo. Este agente es conocedor de todas las unidades que siguen vivas en la partida, de esta forma puede responder a peticiones de solicitud de servicios. Este agente permite a las tropas conocer cuáles de sus compañeros siguen vivos y cuales no para realizar las llamadas de apoyo mencionadas anteriormente.

- **Agente BDITroop:**
  Este agente representa a una tropa de la partida, son agentes híbridos de la arquitectura deliberativa BDI. Los agentes BDITroop pueden realizar algunas acciones como disparar, capturar la bandera, generar paquetes de ayuda, moverse, etc. Este se comunica con el agente service para obtener información acerca del estado de sus compañeros de equipo. Este tipo de agente se divide en tres subclases, que corresponden con los roles que adoptan cada una de las tropas durante la partida, cada una de ellas proporciona un servicio distinto.

    - BDIMedic:
      Corresponde con los médicos, ofrecen el servicio médico, creando paquetes de medicina para aquel agente que lo solicite

    - BDIFieldOp: corresponde con el operador de campo. Proporciona el servicio de munición, permitiendo generar paquetes de munición cerca de los compañeros que lo soliciten.

    - BDISoldiers: son los soldados del equipo, su servicio es ofrece apoyo a un compañero, movilizándose cerca de la posición del agente que lo solicita.

Estos son los tres tipos de agentes principales que participan durante el transcurso de una partida. Es importante mencionar a otro tipo de agente, el agente Pack, que son los paquetes que se crean durante el juego. Estos son: ObjectivePack, que representa la bandera, MedicPack, que representa los paquetes de medicina creados por los médicos y, por último, AmmoPack, que representa los paquetes de munición creados por los operadores de campo. Estos agentes se quedan en una posición estática y desaparecen cuando un agente de tipo BDITroop pasa por la misma posición en la que se encuentra el paquete, o para los paquetes médicos o de munición, agotan un tiempo límite.

Todos los agentes mencionados anteriormente son creados en SPADE, y se comunican entre ellos mediante el sistema de mensajería instantánea.

2.4 Django

Django es un framework web de alto nivel que se ha utilizado durante la primera versión del proyecto. Está desarrollado en Python y diseñado para ofrecer un desarrollo rápido y eficiente de aplicaciones web ofreciendo numerosas ventajas y características que lo convierten en una de las mejores opciones en la actualidad para los desarrolladores. [11]

Django se basa en el patrón de diseño Modelo-Vista-Controlador (MVC) [12]. Esta arquitectura divide la aplicación en 3 partes:

- Modelo: representa la capa de datos de la aplicación, es donde se definen las clases y estructuras de datos que representan los objetos de la aplicación y su interacción con la base de datos. En Django estos modelos se definen utilizando la clase “Model” con sus campos correspondientes. Los modelos se encargan de realizar operaciones de consulta, creación, eliminación y modificación en la base de datos.

- Vista: se encarga de presentar los datos al usuario y maneja la lógica relacionada con la interacción del usuario. En Django, la vista se implementa de forma general mediante una función, que recibe solicitudes HTTP del cliente y devuelve una respuesta. En este caso en la vista se manejan los datos y posteriormente se envían a la plantilla para su representación.

- Controlador (Plantilla): en Django se sustituye el controlador por el *template* o plantilla. La plantilla es una representación visual de cómo se mostrarán los datos. En Django, las plantillas utilizan una sintaxis especial, muy similar al HTML, que permite añadir lógica y variables creando de esta manera un contenido dinámico. Django utiliza su propio motor de plantillas que ofrece características como bucles y condicionales entre otras cosas. Las plantillas se combinan con los datos obtenidos de la vista y se renderizan para producir una respuesta final que es mostrada al usuario.

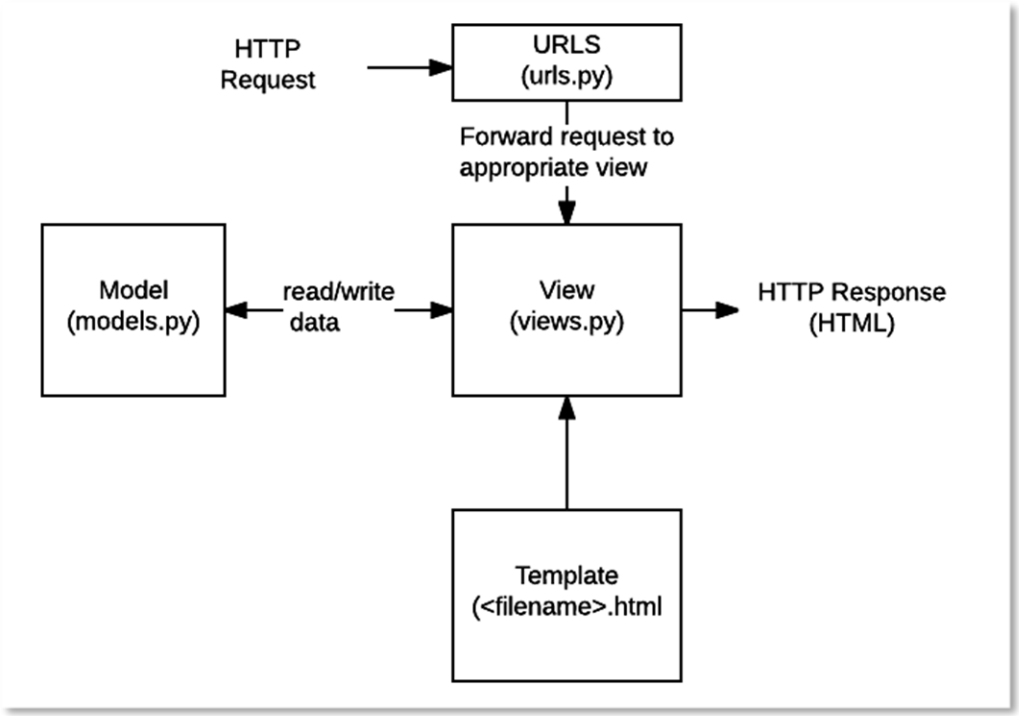


Figura 4. Patrón MVT: Modelo-Vista-Template

Django también ofrece un sistema de enrutamiento de URLs, que permite asociar las URL entrantes en el navegador con las vistas correspondientes que se encargan de manejar dichas solicitudes. El enrutamiento se define en un archivo llamado “urls.py”, como se muestra en la Figura 4. Cada patrón de URL está compuesto principalmente por dos elementos:

- El patrón de la URL, que puede ser una cadena de texto representando la ruta esperada en el navegador o puede contener otras expresiones más complejas muy útiles para patrones URL más flexibles.
- La vista asociada, que es la función encargada de manejar la solicitud.

Cuando la aplicación de Django recibe una solicitud HTTP, el sistema de enrutamiento examina las URL definidas en el archivo urls.py y busca una coincidencia con la URL entrante. Una vez encontrada la coincidencia, pasa el control a la vista asociada a esa URL, normalmente definida en el archivo views.py y encargada de devolver una respuesta.

Una característica importante de Django es la integración del middleware, que es una capa intermedia entre el servidor web y las vistas de la aplicación. Actúa como un conjunto de componentes que procesan las solicitudes HTTP entrantes y las respuestas salientes antes de que lleguen o abandonen la vistas [13]. El middleware se configura en el archivo "settings.py" (archivo donde se configura la aplicación de Django), y se define como una lista ordenada de clases middleware. Django incluye de forma predeterminada un conjunto de middlewares, pero también ofrece la posibilidad a los desarrolladores de crear y personalizar middlewares para implementar funcionalidades adicionales. Durante este proyecto se ha creado un middleware para el manejo de los distintos idiomas que se han introducido en la aplicación.

Otra de las herramientas de Django utilizadas durante el desarrollo de la primera versión del proyecto es el soporte para la internacionalización y localización de aplicaciones web, permitiendo de esta manera adaptar la aplicación a diferentes idiomas. Mas adelante se profundizará acerca de esto y de cómo esta implementado en la aplicación.

3. Análisis y diseño

En este capítulo se proporcionará con detalle la lista de requisitos principales de la nueva versión de pyGOMAS, así como la arquitectura del proyecto y las tecnologías utilizadas.

3.1. Análisis de requisitos

A continuación, se van a describir una serie de requisitos que permitirán a los usuarios mejorar drásticamente el uso de pyGOMAS, mejorando la eficiencia y la eficacia de todas las tareas de configuración y ejecución, además de añadir nuevas opciones al usuario para que puedan visualizar e interactuar con una partida a tiempo real.

- **Ejecución de pyGOMAS**. PyGOMAS debe permitir al usuario el lanzamiento de una partida, dejando de lado la ejecución de comandos en el terminal.
- **Configuración del manager**. Se debe facilitar al usuario la configuración eficiente del agente manager, sin tener que especificar cada uno de los parámetros de configuración en el comando de ejecución.
- **Configuración de las tropas**. Se debe permitir la posibilidad al usuario de generar de forma sencilla e intuitiva el fichero JSON que contiene toda la configuración de los agentes tropa que participarán durante la partida.
- **Guardar y crear partida**. Se debe facilitar al usuario un sistema de guardado de partidas, de forma que no se pierda la configuración de los agentes y permitiendo posteriormente cargar una partida con la configuración de cada uno de los agentes ya definida. Además, al crear una nueva partida, los agentes se autoconfiguran con valores predefinidos para que se pueda lanzar la partida de forma más rápida.
- **Edición/Creación de mapas**. PyGOMAS debe ofrecer una nueva herramienta que permita a los usuarios diseñar de forma libre el terreno de batalla. Esto implica la creación de un mapa desde cero hasta la edición de los mapas ya existentes. También debe ofrecer la posibilidad de asignar un mapa a una partida, para que durante su ejecución el juego transcurra en el terreno de batalla que el usuario seleccione.
- **Sistema multilenguaje**. La interfaz web debe ser capaz de adaptarse a las preferencias lingüísticas de los usuarios, ofreciendo un amplio abanico de distintos idiomas.
- **Ejecución desde distintas maquinas**. Se debe permitir la posibilidad de ejecutar y lanzar agentes en una misma partida desde diferentes máquinas.
- **Monitorización de partida**. PyGOMAS debe permitir la monitorización de la partida a tiempo real, esto implica poder visualizar que agentes tropa que participan en la partida y distinguir cuáles de ellos siguen vivos y cuáles no.
- **Visualización de estadísticas**. El usuario debe poder visualizar las estadísticas generadas por pyGOMAS tras la finalización de una partida.
- **Finalización de partida**. PyGOMAS debe contener un sistema que permita al usuario finalizar una partida en curso, cancelando la sesión de todos los agentes conectados al servidor XMPP que forman parte de la partida.
- **Cerrar aplicación**. Los usuarios deben poder cerrar la interfaz web sin dejar ningún rastro de agentes conectados al servidor XMPP.

3.2. Diseño de la solución

En esta sección, se presenta el diseño de la solución de la aplicación mediante la implementación del patrón arquitectónico Modelo-Vista-Controlador (MVC). Se detallan los componentes clave del diseño, sus interacciones y cómo se alcanzará el objetivo de proporcionar herramientas útiles para los estudiantes.

3.2.1 Arquitectura MVC

Anteriormente se ha explicado en términos generales el patrón Modelo-Vista-Controlador (MVC). En este apartado se detallará las funciones específicas que tiene cada parte del patrón dentro de la aplicación.

3.2.1.1. Modelo

El modelo representa la lógica subyacente y los datos de pyGOMAS. Está diseñado para interactuar directamente con el programa y acceder a los agentes inteligentes proporcionados por SPADE. Entre sus funciones clave podemos encontrar:

- Establecer la conexión con pyGOMAS y sus agentes
- Gestionar la recopilación de datos relevantes de los agentes
- Implementar la lógica de interacción con los agentes pertenecientes a pyGOMAS, para de esta manera, ejecutar acciones específicas.

3.2.1.2. Vista

La vista se encarga de la presentación visual de la información al usuario y de capturar sus interacciones. Esta parte del patrón MVC se encarga de los siguientes puntos:

- Mostrar una interfaz de usuario intuitiva para interactuar con pyGOMAS.
- Presentar datos relevantes en un formato comprensible para el usuario.
- Capturar las acciones del usuario y transmitirlas al controlador para su procesamiento.

3.2.1.3. Controlador

El controlador actúa como intermediario entre el modelo y la vista. Su función principal es gestionar las interacciones del usuario y orquestar las acciones que el modelo debe realizar. En el proyecto, el controlador realiza las siguientes tareas:

- Recibir acciones y eventos del usuario a través de la vista y determinar las acciones correspondientes en el modelo.
- Comunicar las solicitudes y comandos al modelo para que interactúe directamente con pyGOMAS.
- Actualiza la vista con los cambios en los datos y la información proporcionada por el modelo.

3.2.2 Interacciones y flujo de trabajo

El flujo de trabajo general del proyecto basado en el modelo MVC será el siguiente:

1. El usuario interactúa con la vista al realizar acciones en la interfaz de usuario
2. El controlador recibe las peticiones y las traduce en comandos o solicitudes para el modelo.
3. El modelo interactúa con los agentes de pyGOMAS o realiza otras acciones.
4. El modelo actualiza sus datos y notifica al controlador sobre los cambios.
5. El controlador actualiza la vista proporcionando al usuario la información adecuada.

3.2.3 Beneficios del diseño

El uso del patrón MVC ofrece los siguientes beneficios:

- Separación de preocupaciones. Cada componente tiene un propósito y responsabilidades claras, lo que facilita la colaboración y el mantenimiento del código.
- Reutilización de código. La estructura modular permite reutilizar componentes en futuros proyectos o mejoras.
- Escalabilidad. Si pyGOMAS se expande en un futuro, la arquitectura MVC facilita la incorporación de nuevas funcionalidades.

3.3. Tecnologías utilizadas

En este apartado se describirán todas las tecnologías utilizadas durante el desarrollo del proyecto.

- **Python**[1]. Es un lenguaje de programación interpretado y versátil, conocido por su sintaxis legible y fácil de comprender. Se destaca en una amplia gama de aplicaciones, desde desarrollo web y científico hasta automatización y aprendizaje automático. Su comunidad activa, bibliotecas y enfoque en la legibilidad hacen de Python una opción popular para programadores de todos los niveles.
- **HTML, CSS y JavaScript**[2]. Son las tecnologías fundamentales para el desarrollo frontend en la web:
    - **HTML**. Es el lenguaje de marcado utilizado para estructurar y organizar el contenido de una página web. Define la jerarquía de los elementos y su contenido, como el texto, imágenes y enlaces.
    - **CSS**. Se utiliza para dar estilo y diseño a las páginas web. Define cómo se ven los elementos HTML, incluyendo colores, fuentes, espaciado y disposición.
    - **JavaScript**. Es un lenguaje de programación que se ejecuta en el navegador del usuario. Permite crear interacciones dinámicas en una página web, como animaciones, validaciones de formularios y actualizaciones en tiempo real.
- **SPADE** (Smart Python Agent Drevelopment Environment). Es una plataforma de sistemas multiagente desarrollada en Python. Proporciona herramientas y bibliotecas que

[1] https://es.python.org/
[2] https://developer.mozilla.org/es/docs/Web/JavaScript

facilitan el modelado, comunicación y simulación de agentes inteligentes que interactúan y cooperan para lograr objetivos individuales y colectivos.

- **Anaconda3**[3]**.** Es una plataforma que facilita la creación y administración de entornos virtuales en Python. Permite a los usuarios crear ambientes aislados para desarrollar y ejecutar proyectos con paquetes y versiones específicos, simplificando la gestión de dependencias y garantizando la coherencia en distintos proyectos.
- **Django.** Es un framework web de alto nivel y de código abierto escrito en Python. Proporciona herramientas y estructuras para simplificar la creación de aplicaciones web robustas y escalables. Con características como un ORM (Mapeo Objeto-Relacional), administración de bases de datos, autenticación y enrutamiento. Django agiliza el desarrollo al seguir el patrón Modelo-Vista-Controlador (MVC) y promover buenas prácticas de seguridad y rendimiento.
- **Jinja2**[4]. Es un motor de plantillas para Python que permite la generación dinámica de contenido HTML, XML, JSON y otros formatos de texto. Utilizado principalmente en el contexto de aplicaciones web construidas en frameworks como Flask y Django, Jinja2 facilita la separación de la lógica y la presentación al permitir la inserción de variables y estructuras de control en las plantillas, lo que simplifica la creación de interfaces dinámicas y personalizadas.
- **Selenium**[5]. Es una librería que permite interactuar con navegadores web de forma programática. Utilizada en Python y otros lenguajes, facilita la creación de scripts que simulan acciones humanas en páginas web, útil para pruebas funcionales y control de calidad en el desarrollo web.
- **AJAX** (Asynchronous JavaScript and XML). Es una técnica de programación web que permite actualizar partes específicas de una página sin tener que recargarla por completo. Utiliza tecnologías como JavaScript y XML para realizar solicitudes y recibir respuestas del servidor de forma asíncrona. Esto permite una experiencia de usuario más fluida al cargar y mostrar datos en segundo plano, sin interrumpir la interacción del usuario con la página. [14]
- **Visual Studio Code**[6]. Es un editor de código fuente altamente popular y gratuito desarrollado por Microsoft. Ofrece un entorno ligero y extensible con soporte para una amplia gama de lenguajes de programación. Con funciones como resaltado de sintaxis, depuración integrada, control de versiones y una gran cantidad de extensiones, se ha convertido en una elección popular entre los desarrolladores para la edición y desarrollo de software.
- **GitHub**[7]. Es una plataforma de alojamiento en línea que utiliza Git para el control de versiones colaborativo. Permite a los desarrolladores trabajar juntos en proyectos, gestionar cambios, colaborar en código y rastrear problemas, facilitando el desarrollo de software de manera eficiente.

[3] https://www.anaconda.com/
[4] https://jinja.palletsprojects.com/en/3.1.x/
[5] https://www.selenium.dev/
[6] https://code.visualstudio.com/
[7] https://github.com/

4. Desarrollo de la solución propuesta (Primera versión)

A lo largo de este apartado se explicará detalladamente como se ha configurado el proyecto utilizando Django y como han sido implementadas las distintas funcionalidades para satisfacer todas las necesidades de los usuarios, mencionadas en el apartado 3.1.

Tras evaluar los principales objetivos y los requisitos de la nueva versión de pyGOMAS, se optó en utilizar Django como primera opción. Ya que, Django ofrece características muy útiles que permiten la creación de aplicaciones web, destacando entre ellas el uso de plantillas, que facilitan la generación de contenido dinámico y la presentación de datos en HTML de forma modular y reutilizable, y un sistema de enrutamiento de URLs que mapea las solicitudes de los usuarios a las vistas correspondientes.

Antes de comenzar con la configuración de Django y el desarrollo de las nuevas funcionalidades, se prepara un entorno de trabajo para que esto sea posible. Primero se instala pyGOMAS en un entorno virtual, utilizando Anaconda3 como herramienta para la gestión de paquetes y entornos virtuales e instalando la versión 3.7 de Python, que es la versión con la actualmente está desarrollado pyGOMAS. Una vez este el entorno virtual preparado se procede con la instalación de Django.

4.1 Configuración del proyecto con Django

Tras la instalación de Django en el entorno virtual, se define una estructura de proyecto que sigue el modelo estándar de Django. A continuación, se nombran los archivos más relevantes de la aplicación:

- **Manage.py:**
  Este archivo es el punto de entrada principal para la administración de la aplicación Django. Se utiliza para ejecutar comandos de gestión, como iniciar el servidor de Django. Este archivo no ha sido modificado durante el desarrollo, pero es esencial para el funcionamiento de la aplicación.

- **Settings.py:**
  En este archivo se encuentra la configuración principal de la aplicación. Algunas de las más relevantes añadidas para la realizar este proyecto han sido:

  - Configuración de la ruta para la carpeta “static”: es muy común el uso de recursos estáticos durante el desarrollo de la aplicación, entre ellos destacan los archivos de estilos CSS, los archivos de JavaScript y distintos iconos o imágenes. Django nos proporciona herramientas para poder utilizar estos recursos tras ejecutar la aplicación en el servidor añadiendo la ruta de la carpeta “static” a la configuración. En este caso se ha definido como ruta a los recursos estáticos **BASE_DIR + pygomasApp/static**, donde BASE_DIR

corresponde con la dirección base donde está instalada la aplicación, también configurable desde settings.py.

- Configuración de la carpeta "Locale": se plantea añadir un sistema multilenguaje de forma que se pueda cambiar el idioma de la aplicación. Para ello se añade una nueva carpeta "Locale" al proyecto y se configura la ruta desde el archivo settings.py. En este caso la ruta añadida es similar a la anterior: **BASE_DIR + pygomasApp/Locale.** Mas adelante se profundizará más acerca del contenido de esta carpeta y su funcionamiento.

- Middlewares: además de los middlewares que Django usa por defecto, como *SecurityMiddleware*, *LocaleMiddleWare* o *MessageMiddleware*, se ha añadido un middleware personalizado para el desarrollo del sistema multilenguaje comentado en el punto anterior. Más adelante se especificará el funcionamiento de este middleware personalizado cuyo nombre es "LenguageMiddleware**"**.

- **Urls.py:**
  En este archivo se encuentran definidas todas las URLs que forman parte del mapa de navegación de la aplicación. Para cada URL definida, se inserta el nombre de la función controladora, que se ejecutara cada vez que el usuario acceda a la URL en específico. En la Figura 5 se muestra el mapa de navegación completo de la primera versión del proyecto, cada bloque tiene asociada una URL además de su función controladora.

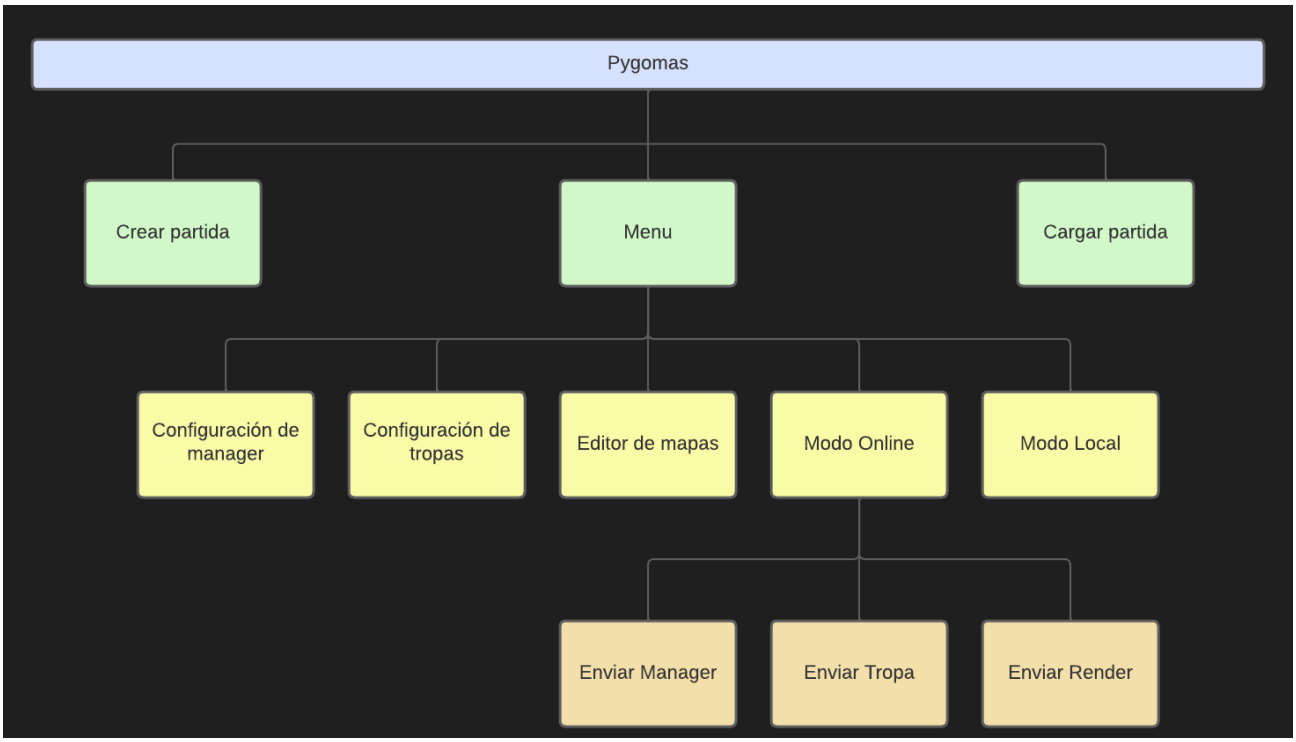


Figura 5. Mapa de navegación de pyGOMAS

- **Views.py:**
  En este archivo están definidas todas las funciones controladoras. Para cada vista de la aplicación se ha definido una función distinta. Además, estas funciones o controladores manejan tanto las peticiones de tipo GET como las peticiones de tipo POST. Todas las funciones tienen como parámetro de entrada una instancia de la clase "HttpRequest", que representa la solicitud HTTP, y devuelven una respuesta a dicha solicitud. Principalmente se han utilizado dos funciones para lograr esto

    - Función Render: esta función renderiza una plantilla HTML y la devuelve como respuesta a la solicitud HTTP. Es muy útil ya que permite pasar como parámetro el objeto *request*, que es la instancia de la clase HttpRequest y representa la solicitud HTTP inicial. También se especifica en los parámetros la plantilla HTML a renderizar. Por último, pero no menos importante, como parámetro opcional se puede añadir un diccionario de Python, donde podemos definir algunas variables con su respectivo valor que permitirá crear un contenido dinámico en las vistas mostradas al usuario.

    - Función Redirect: función utilizada principalmente para navegar a través de la aplicación, sirve para redirigir la petición HTTP a una URL específica, esta URL se le pasa a la función Redirect como parámetro de entrada. Cuando la función se llama, el servidor envía una respuesta HTTP al cliente con la nueva URL a la que se debe redirigir. El cliente, al recibir esta respuesta, realizará una nueva solicitud a la URL de destino y mostrará la página correspondiente.

  En el archivo views.py también están definidas algunas variables globales, que permiten almacenar los datos durante la ejecución de la aplicación.

4.2 Funcionalidades

El objetivo del proyecto consiste en añadir funcionalidades para facilitar a los usuarios la configuración y el lanzamiento de pyGOMAS, además de permitir la intervención del usuario durante la ejecución de una partida. A lo largo de este apartado se explica detalladamente cada funcionalidad, el motivo de su implementación, que problemas soluciona y como están implementadas en el proyecto, añadiendo algunos ejemplos gráficos.

4.2.1 Ejecución de pyGOMAS

La antigua versión de pyGOMAS no dispone de una interfaz de usuario que facilite la ejecución de las partidas. Es por ello por lo que la primera de las funcionalidades propuestas consiste en diseñar una interfaz simple para poder configurar los agentes necesarios para que la partida de inicio. Estos agentes son el manager, las tropas y el render.

Para poder iniciar una partida en la versión antigua era necesario lanzar estos tres agentes en terminales separadas. Para ello se ejecutaban los siguientes comandos:

- pygomas manager -j <nombreManager>@<servidorXMPP> -m <mapa> -sj <nombreService>@<servidorXMPP> -np <NumeroDeTropas>
- pygomas run -g <ficheroAgentes.json>
- pygomas render

Esto era muy poco práctico por muchas razones, entre ellas, definir cada vez el agente manager con todas sus posibles configuraciones, crear un archivo JSON donde se definen todos los agentes tropa y sus características, y ejecutar cada vez el comando para lanzar el visualizador de la partida.

Para solucionar estos problemas se plantea crear una interfaz para el usuario donde le permita configurar todos los agentes, y después, con solamente un clic, poder lanzar la partida de una forma más rápida y cómoda.

Para el manager se crea un formulario donde se pueda introducir todos los datos que afectan a la configuración del agente, permitiendo, de esta manera, una máxima flexibilidad a la hora de crearlo. Además, el proceso de lanzamiento es más intuitivo, ya que se dejan de lado los parámetros de configuración en los comandos, como -mp, -np, -j o -sj.

Figura 6. Configuración del agente Manager

Tras darle a guardar, los valores quedan almacenados en un archivo de texto, esto forma parte de otra funcionalidad que se comentará más adelante.

La configuración de los agentes tropa se especifica en un archivo JSON. El archivo contiene cinco parámetros:

- Host: guarda el nombre del servidor XMPP
- Manager: guarda el nombre del manager de la partida, para lanzarlo en local, deberá contener el valor que le hemos dado al configurar el manager

- Service: guarda el nombre del agente service, deberá contener el valor introducido durante la configuración del manager.
- Axis: es una lista de las tropas que pertenecen al equipo Eje. En cada elemento de la lista se especifica el nombre de la tropa, el rango (médico, operador de campo o soldado) y una contraseña para conectarse al servidor XMPP. También existe la posibilidad de añadir un parámetro extra al agente tropa, que consiste en la cantidad de agentes que deseamos crear, de esta manera si necesitamos crear 2 agentes iguales podemos poner la etiqueta "amount: 2", esto lanzará dos agentes con la misma configuración modificando automáticamente el nombre, ya que no es posible que dos agentes tropa se llamen igual.
- Allied: es una lista al igual que Axis, aquí se especifican los agentes tropa pertenecientes al equipo Aliado. Los parámetros de cada elemento de la lista son los mismos que para los elementos de Axis.

La única manera de lanzar los agentes tropa era creando un archivo JSON incluyendo todos los parámetros anteriores o modificar uno ya existente, siendo una forma muy poco practica para el usuario. Se decidió implementar una nueva vista que te permitiera autogenerar el archivo JSON de forma automática. Para ello se ha desarrollado una vista que permita al usuario seleccionar el número de jugadores por equipo que se desea y, posteriormente, añadir el nombre y el rango del agente, como se puede observar en la Figura 7.

Figura 7. Configuración de agentes tropa

Tras pulsar el botón guardar, en el directorio pygomas/tropas se creará un archivo JSON rellenando los parámetros con los valores introducidos. Para los parámetros Host, Manager y Service, recogerá los valores de la configuración del Manager.

Una vez el agente manager y las tropas estén configurados a través de la interfaz, ya es posible realizar un lanzamiento de una partida. Cabe destacar que esta partida se ejecutara en local, es decir, todos los agentes son lanzados desde la misma máquina. Para poder iniciar la partida se ha diseñado una pequeña interfaz donde se podrá elegir cuales de los agentes queremos lanzar, como se muestra en la Figura 8.

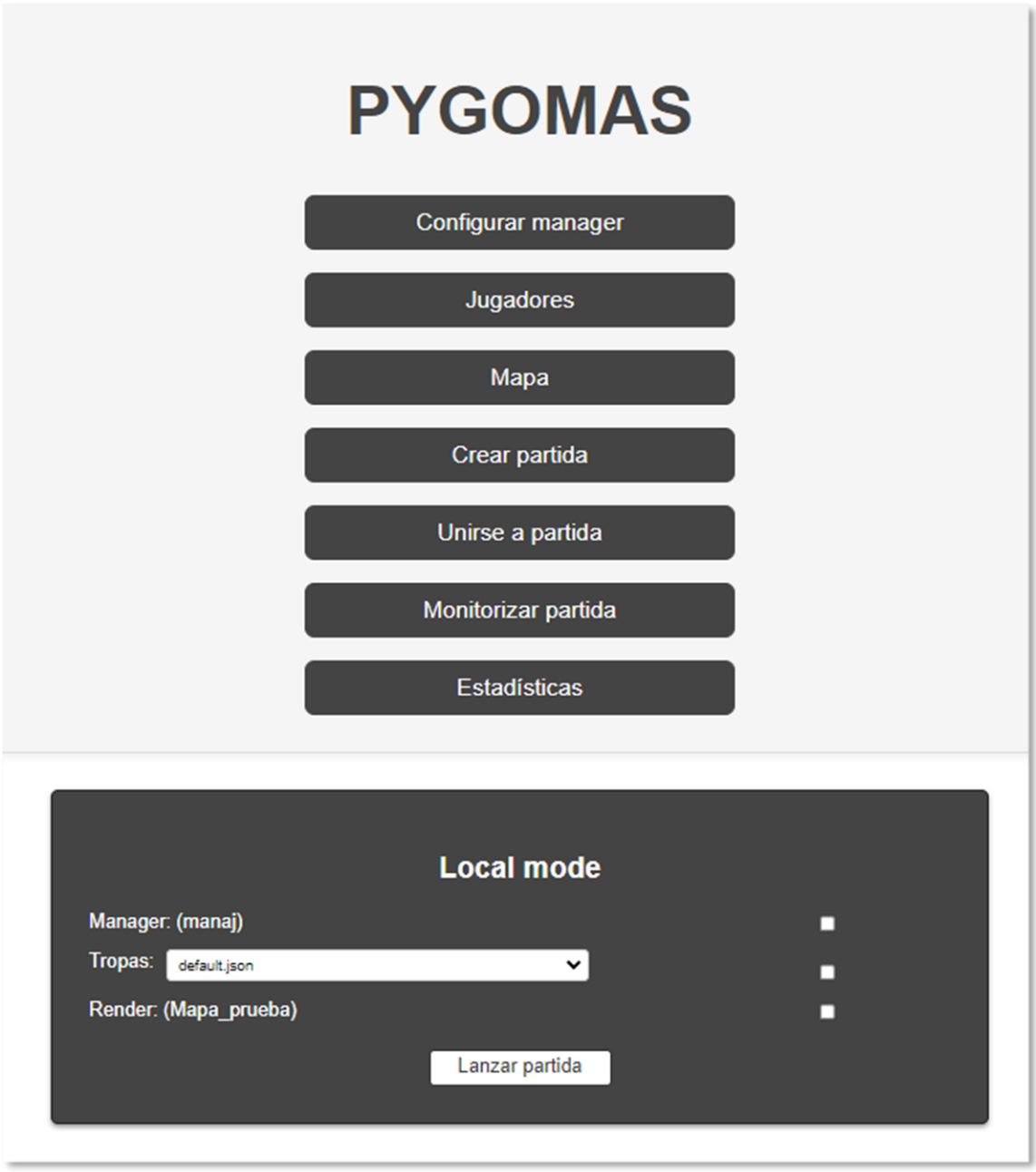


Figura 8. Menú principal con opción de 'Crear partida' seleccionada

En la figura 8 se puede observar el menú principal de la aplicación con la opción de crear partida seleccionada. En este menú hay más opciones que permiten visualizar la configuración del manager y la configuración de los agentes tropa vistos en las Figuras 6 y 7. Las opciones de 'Mapa' y 'Unirse a partida' son funcionalidades que se explicaran más adelante. Para poder ejecutar una partida sería suficiente con seleccionar los agentes a lanzar, además se muestra información del nombre de los agentes y del mapa.

Se puede observar que no se dispone de ninguna vista la configuración del render, ya que su única configuración es el nombre del manager que lanzara la partida, este valor se introduce

automáticamente obteniendo el valor de la configuración del manager, de forma muy similar a los parámetros manager y service del archivo JSON de las tropas.

Una vez hagamos clic en 'Lanzar partida' se ejecutarán en terminales distintas los comandos respectivos a la ejecución de cada agente visto anteriormente, y tras unos segundos, la partida se iniciará.

4.2.2 Crear nueva partida y cargar partida

Tras mejorar la configuración de los agentes y el lanzamiento de las partidas, se plantea una nueva mejora para aumentar la rapidez a la hora de ejecutar varias partidas seguidas. Es cierto que configurar tanto el manager como las tropas se ha vuelto una tarea más sencilla, así como poder lanzarlos con un solo clic, dejando de lado la ejecución de los comandos, pero todo el proceso de configuración era obligatorio para poder lanzar una partida, por ello se propone un diseño que permita a los usuarios guardar todas las configuraciones y luego poder cargarlas, similar al sistema de guardado de partidas de cualquier videojuego.

Para empezar con el desarrollo primero había que pensar una forma en la que la configuración de los agentes quedara guardada en el proyecto. Como no se dispone de una base de datos, se plantea crear una carpeta, en este caso llamada "Saves", donde se almacenarán archivos de texto. Cada archivo de texto tendrá el nombre que el usuario introduzca, y el contenido de cada uno de ellos será la configuración que una partida necesita para su ejecución. De esta forma, todos los archivos contendrán la siguiente información:

- SERVER: el nombre del servidor XMPP, se utiliza para configurar el manager y el parámetro "Host" del JSON
- NUM_PLAYERS: la cantidad de tropas totales que participan en una partida. Es un valor obligatorio para ejecutar el manager.
- MAP_NAME: es el nombre del mapa o terreno de guerra definido en la carpeta "Maps", más adelante se entrará más en detalles acerca de esto.
- AGENTS: es el nombre del JSON donde están configuradas las tropas.
- MANAGER: es el nombre o JID del agente manager
- SERVICE: es el nombre o JID del agente service, lanzado por el manager.
- PORT: puerto donde son lanzados los agentes.
- TIME: es el tiempo de duración de la partida.

Para proporcionar al usuario una forma rápida y sencilla, se ha diseñado una nueva vista antes del menú principal, mostrado en la Figura 8. En esta vista, el usuario puede seleccionar si desea cargar una partida ya existente o, por lo contrario, desea crear una nueva.

Figura 9. Menú para crear una nueva partida

En la Figura 9 se muestra como el usuario, tras escoger la opción “Nueva partida”, puede insertar un nombre que identifique a dicha partida. Una vez haga clic en “crear”, se navegará directamente el menú principal, creando durante el proceso un nuevo archivo de configuración con el nombre introducido por el usuario y unos valores de configuración por defecto para que sea posible ejecutar la partida de forma directa, sin necesidad de configurar los agentes. Cuando el usuario ejecute la partida, los datos se leerán directamente en el archivo de texto que corresponde con la partida en la que nos encontramos actualmente. Además, en el menú principal, se muestran los datos clave para que el usuario pueda consultar cómodamente en que partida se encuentra, el nombre del archivo JSON asignado a dicha partida y el mapa.

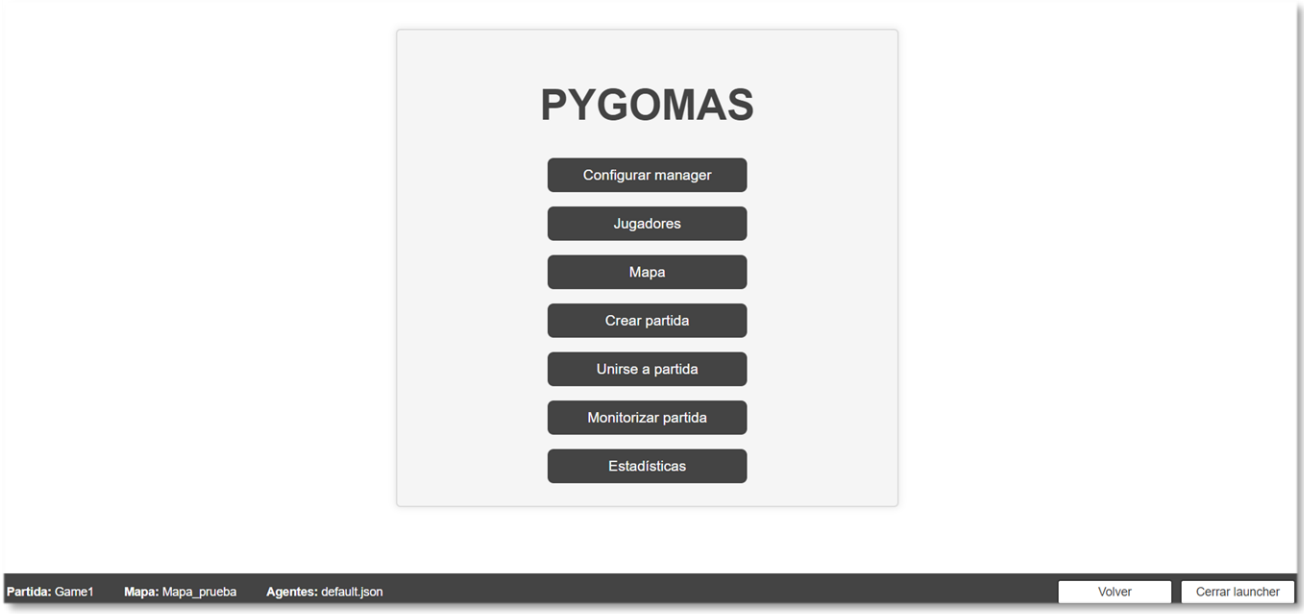


Figura 10. Menú principal con los datos clave de la partida

El usuario podrá modificar las configuraciones que necesite, como se ha explicado en el apartado 4.2.1. Estas modificaciones se verán reflejadas en el archivo de configuración respectivo a la partida. Permitiendo, de esta manera, volver a utilizar las mismas configuraciones cuando el usuario las necesite mediante la opción "Cargar partida", como se muestra en la siguiente imagen:

Figura 11. Menú para cargar una partida

Cada partida mostrada en la lista tiene su propia configuración, esto está pensado para que los usuarios puedan crear distintas pruebas con configuraciones completamente distintas, y puedan ir cambiando sin necesidad de reconfigurarlas cada vez, dándole una nueva mejora a la usabilidad de pyGOMAS.

4.2.3 Editor de mapas

Como se ha mencionado anteriormente, las partidas de pyGOMAS transcurren en un terreno de batalla. En este terreno o mapa, se incluyen cuatro elementos:

- Wall: es un elemento sólido, las tropas no pueden atravesarlo.
- Flag: representa el objetivo que debe defender el equipo Eje y que los Aliados deben capturar.
- Axis Spawn: es el lugar donde nacen los agentes tropa del equipo Eje.
- Allied Spawn: zona donde nacen las tropas del equipo Aliado.

PyGOMAS interpreta los mapas leyendo dos archivos de texto, que contienen la información necesaria para crear un mapa con todos los elementos comentados anteriormente.

El primer archivo contiene una representación del mapa mediante el carácter "*", donde cada "*" representa un elemento "Wall". En la Figura 12 se muestra el contenido del archivo "map_03_cost.txt", donde map_03 es el nombre del mapa que se pasa como parámetro al manager.

Figura 12. Ejemplo de archivo _cost.txt

En los archivos _cost.txt, solo se definen los elementos “Wall”, pyGOMAS interpreta la posición del resto de elementos mediante el otro archivo, en este caso sería el archivo map_03.txt. El contenido de este archivo se basa en indicar la posición exacta de los elementos restantes mediante coordenadas. Para indicar la posición del elemento “flag” se introducen dos números que representan las coordenadas donde estará situada la bandera durante la partida. Los elementos “spawn” para ambos equipos son algo diferentes, ya que, es necesario introducir dos coordenadas para cada base de cada equipo, la primera coordenada indica la esquina superior izquierda, y la segunda coordenada la esquina inferior derecha. Con estas dos coordenadas se puede formar un área, que será la zona donde nacen los agentes tropa.

Teniendo en cuenta esto, se puede entender la dificultad y el tiempo que requiere personalizar un mapa sin ningún tipo de herramienta, por lo que ha implementado una vista para facilitar al usuario esta tarea.

Primero, había que pensar una forma de poder representar visualmente un mapa al usuario. Teniendo en cuenta que el mapa tiene un tamaño de 32x32 y que en los archivos se indican coordenadas exactas para definir la posición de los elementos, una matriz de 32x32 parecía la mejor solución para su representación. Además, cada posición de la matriz contendría un valor que representara su contenido. Para poder lograr esto se definieron dos clases en Python:

- Clase Box:
  Representa una posición exacta en el mapa, tiene 4 atributos tipo Bool: *hasFlag*, *hasWall*, *hasAxisSpawn* y *hasAlliedSpawn*. La idea es que cualquier instancia de Box solo pueda tener uno de los 4 atributos con el valor ‘True’, representando el contenido de una coordenada en específico. En caso de que todos los atributos se encuentren en false, significa que esa coordenada está vacía y no dispone de ningún elemento del mapa. Además, la clase Box, tiene implementadas algunas funciones muy útiles como las funciones *get* y *set*, útiles para recuperar el valor de los atributos y modificarlos, y una función *clear*, que asigna a ‘False’ todos los atributos de la instancia.
- Clase Map:
  Esta clase representa el mapa completo. Como único atributo tiene una matriz de 32x32, donde cada elemento de la matriz es una instancia Box. Al crear una instancia de tipo Map, por defecto, los borden de la matriz, son objetos Box con el atributo hasWall = True, definiendo el borde del mapa. Todo el interior de la matriz son objetos Box con los 4 atributos con el valor de ‘False’. La clase Map también dispone de funciones útiles para el desarrollo, como, por ejemplo, la función *getBox*, que pasando como parámetros unas coordenadas devuelve la instancia Box que se encuentra en dicha posición, funciones tipo *set*, para modificar el contenido de una coordenada exacta, una función llamada *createTxt*, que mediante una instancia de Map, construye los dos archivos de texto necesarios para que pyGOMAS puede interpretar el mapa, o también el proceso inverso, con la función *loadMap*, que construye un mapa a través de la lectura de los dos archivos de texto. También se pueden encontrar funciones que devuelven las coordenadas de los elementos flag, axisSpawn y alliedSpawn.

Una vez diseñado el modelo para representar todos los mapas, se muestra al usuario gráficamente, a través de una vista HTML, el resultado final es el siguiente:

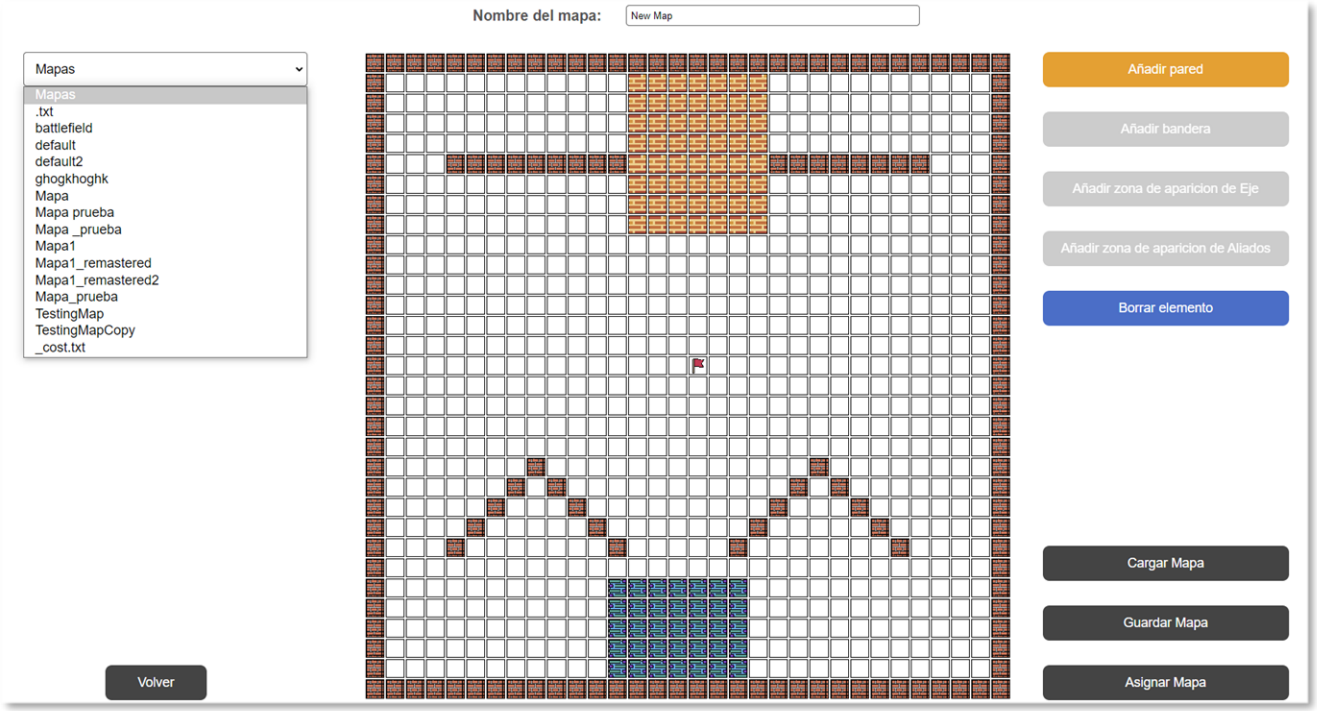


Figura 13. Editor de mapas

En la Figura 13 se muestra el editor de mapas completo, con todas las herramientas que ofrece al usuario, entra las que podemos encontrar las siguientes:

- Modificar mapa: permite al usuario añadir y eliminar elementos en el mapa. Mediante los botones azules de la Figura 13, el usuario puede añadir paredes, una bandera y una zona de spawn para cada equipo, limitando tanto la bandera como las zonas de spawn, ya que no pueden existir más de una en cada mapa. Para limitar el número de elementos al usuario, los botones se desactivan una vez los elementos hayan sido incluidos en el mapa. Si el usuario desea borrar uno de los elementos limitados, el botón quedará activo nuevamente para que se pueda insertar nuevamente el elemento correspondiente. Las casillas azules y rojas corresponden con las zonas de spawn de ambos equipos, para poder añadir una zona, bastaría con hacer clic en el botón y después seleccionar dos casillas del mapa. Automáticamente se creará un área donde se limita la zona de aparición. La bandera es el punto verde, es tan simple como hacer clic en una casilla con el botón "Añadir bandera" seleccionado. Las parades funcionan igual que la bandera, con la diferencia de que no existe ningún límite para añadirlas.
- Cargar mapa: permite visualizar en la tabla un mapa ya existente. A través del selector, que muestra una lista completa de todos los mapas que se encuentran en la carpeta 'Maps', el usuario podrá seleccionar uno de los mapas y después darle al botón 'Cargar mapa', que, utilizando la función *loadMap* mencionada anteriormente, se creará una instancia Map, y se cargará en la tabla, mostrando de forma visual el aspecto del mapa. Tras la carga del mapa, el usuario puede modificarlo libremente.

- Guardar mapa: el usuario podrá guardar el diseño de un mapa, permitiéndole el uso de este en cualquier otro momento. Cuando se guarda un mapa, se crean los 2 archivos de texto a través de la función *createTxt* y se guarda en la carpeta 'Maps' una subcarpeta con el nombre del mapa introducido por el usuario. Esta subcarpeta contendrá los archivos de texto necesarios para que pyGOMAS pueda interpretar los mapas.
- Asignar mapa: como su nombre indica, asigna el mapa en la partida actual. En el apartado 4.2.2 se habla sobre los archivos de configuración que tiene cada partida. Tras asignar un mapa, se modifica la configuración de 'MAP_NAME', con esto conseguimos que al ejecutar la partida se pase como parámetro el nombre del mapa que acabamos de asignar. Para poder lanzar una partida con un mapa en específico, este debe existir en la carpeta 'Maps', en caso de asignar un mapa sin que este guardado, primero lo guardará y luego lo asignará.

4.2.4 Sistema multilenguaje

Añadir un sistema multilenguaje en cualquier aplicación permite que las personas de diferentes países y culturas puedan usarla en su idioma nativo. Esto hace que la aplicación sea más accesible y fácil de usar para una audiencia global. Además, abrirse a nuevos idiomas y mercados aumenta las oportunidades de negocio y el crecimiento de la aplicación. Al ofrecer opciones de idioma, se muestra respeto por la diversidad y se fomenta una mayor inclusión. Por todas estas razones, se ha implementado un sistema multilenguaje en pyGOMAS durante esta primera versión.

Django ofrece herramientas para implementar todos los idiomas deseados en una aplicación. Para ello se hace uso de archivos de traducción, donde se almacenan las cadenas de texto en diferentes idiomas. Al utilizar funciones y etiquetas específicas de Django, se puede acceder a estas traducciones y presentar el contenido adecuado según el idioma seleccionado por el usuario.

El sistema está implementado a través de archivos .mo y .po, que almacenan cadenas de texto y son generados automáticamente mediante la herramienta 'gettext'. Las cadenas de texto se obtienen de forma automática cuando se encuentra la etiqueta {% trans %} en la plantilla. Después de que los archivos sean autogenerados, se permite modificarlos para agregar las cadenas de texto traducidas, lo que permite cualquier nuevo idioma con mucha facilidad.

Un problema que surgió durante el desarrollo surgió debido a la inexistencia de la base de datos, ya que Django utiliza un middleware ya implementado para detectar en que idioma está la aplicación, obteniendo el valor de la base de datos, pero en este caso, al no disponer de una base de datos, no era posible intercambiar entre varios idiomas.

Para solucionar esto, se desarrolla un nuevo middleware llamado "LenguageMiddleware.py", que se ejecuta antes de que la vista correspondiente al enrutamiento de la URL haya sido llamada, con la idea de que el middleware compruebe que idioma está seleccionado para aplicar la traducción correspondiente. El valor del idioma guardado se obtiene a través de una cookie llamada 'Idioma'.

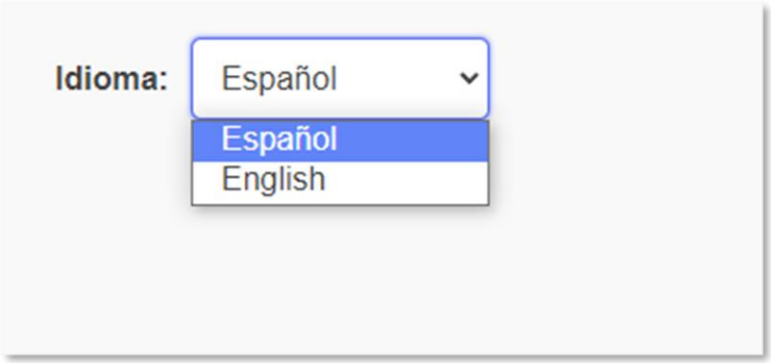


Figura 14. Selector de idiomas

En la Figura 14 se visualiza la plantilla para seleccionar un idioma, en este caso solo se han implementado dos, pero es necesario recalcar que añadir cualquier idioma sería una tarea muy sencilla, ya que solo hay que autogenerar los archivos .po y .mo y luego añadir las traducciones.

Esta plantilla de selección de idioma se puede encontrar en todas las demás vistas, utilizando el sistema de plantillas flexible y modular que ofrece Django, a través de la etiqueta {% block content %}. Esto permite diseñar, en este caso, la plantilla de selección de idiomas e insertarla cómodamente en cualquier otra plantilla de la aplicación. Evitando de esta forma añadir el selector de idiomas a cada una de las plantillas.

Cuando el usuario selecciona un idioma, se guardará en la cookie ‘Idioma’ el valor del idioma seleccionado, tras realizar esta tarea, el middleware se encargará de leer y aplicar el cambio de idioma ante cualquier solicitud.

5. Desarrollo de la solución propuesta (Segunda versión)

Durante este punto del desarrollo, se planteó diseñar un agente global que se pudiese comunicar con los otros agentes que participan durante una partida de pyGOMAS, como el manager, el service, o los agentes tropa.

Como se ha mencionado en el apartado 2.2, el módulo web de SPADE nos permite crear aplicaciones propias servidas por los propios agentes, pudiendo registrar nuevas rutas en el módulo e implementar controladores, siguiendo el paradigma modelo-vista-controlador, una estructura muy similar a la utilizada actualmente con la aplicación de Django. La idea era que el nuevo agente global, al que nos referiremos como 'Launcher', fuese el lanzador de la aplicación.

El principal cambio que dio pie a realizar esta segunda versión fue la adaptación de todo el proyecto para que funcionase con el módulo web del agente Launcher. Esta adaptación entraba en conflicto con Django, ya que la aplicación se estaba ejecutando en el servidor ofrecido por Django, cuando el nuevo objetivo es que se ejecutase a través del módulo web del nuevo agente global. Por este motivo, se decidió eliminar Django del proyecto y seguir trabajando con el módulo web del agente.

5.1 Configuración del agente Launcher

Tras eliminar Django, es necesario reconfigurar el proyecto. Algunos de los archivos de configuración mencionados en el apartado 4.1 quedaron inservibles y había que realizar una adaptación para no perder ninguna funcionalidad.

Para realizar dicha adaptación se crea un nuevo archivo 'launcher.py', el cual crea un nuevo agente SPADE y lo configura para que la aplicación se inicie manteniendo todas las funcionalidades de la primera versión. Para ello se han añadido las siguientes implementaciones que sustituyen los archivos de configuración:

- **Urls.py**
  Este archivo de Django se encargaba de definir todas las URL que formarían parte del mapa de navegación de la aplicación. SPADE proporciona un sistema para registrar URLS siguiendo el paradigma de MVC, replicando el funcionamiento que ya teníamos anteriormente. Para ello se utiliza las funciones *add_get* y *add_post* del módulo web del agente launcher. Estas funciones reciben como parámetros la URL que se quiere introducir en la aplicación, el archivo .html y el nombre de la función controladora. Cabe destacar que las peticiones son tratadas de forma distinta dependiendo del su tipo, GET o POST. En el proyecto se utiliza el mismo controlador para los dos tipos de peticiones. También es posible indicar en los parámetros de estas funciones que no se devuelva ninguna vista, permitiendo la implementación de funciones que solamente devuelvan datos de interés.

- **Views.py**
  Es el archivo donde se encontraban todas las funciones controladoras para cada vista. Dado que Django ofrecía un sistema de herencia de plantillas, permitiendo introducir bloques de contenido html dentro de otras plantillas, donde cada uno de estos bloques tenía su propia función controladora, tras eliminar Django se ha perdido este funcionamiento, obligando a unificar distintos archivos .html que antes estaban separados, lo que conlleva a unificar de la misma forma algunas funciones controladoras que antes estaban separadas y también modificar considerablemente el mapa de rutas de la aplicación, dando como resultado el siguiente mapa:

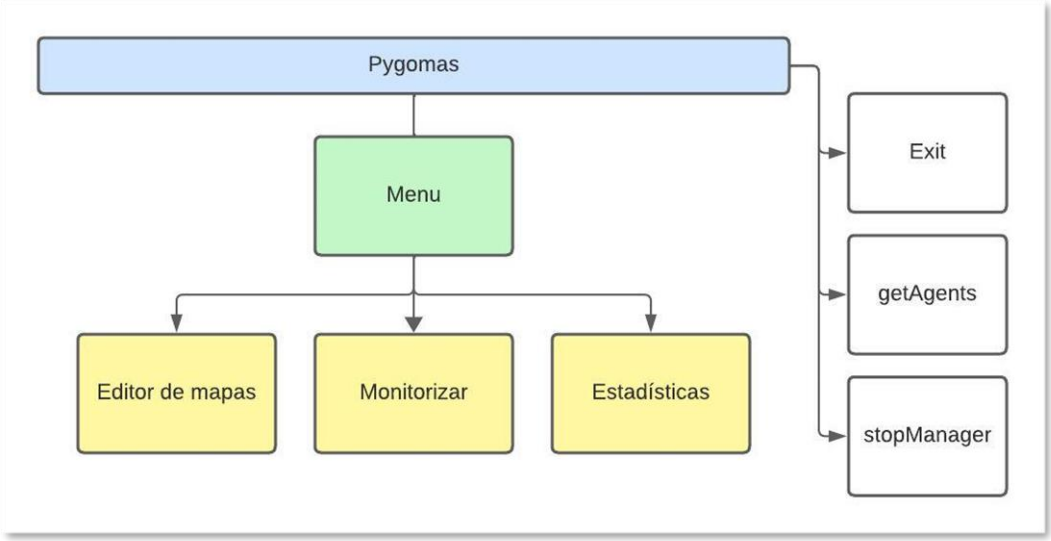


Figura 15. Mapa de navegación de pyGOMAS (segunda versión)

Como se puede observar, el mapa de navegación se simplifica notablemente comparándolo con el de la figura 5. Todas las rutas que ya no se observan en el mapa se han unificado con el menú, lo que ha implicado unificar los archivos .html y los controladores. En las rutas "exit", "getAgents" y "stopManager" no se encuentran vistas, son URLs definidas para cerrar la aplicación, detener una partida y obtener datos sobre los agentes. Se entrará más en detalle acerca de estas funcionalidades más adelante.

- **Settings.py:**
  Es el archivo de Django que contenía toda la configuración de la aplicación, entre ellas las más destacadas eran la ruta de archivos estáticos que se utilizaban en el servidor de Django y la configuración de middlewares junto con el directorio Locale, utilizados para la implementación del sistema multilenguaje.

  Para añadir los archivos estáticos al nuevo servidor creado por el agante Launcher se ha utilizado la función "add_static" proporcionada por el módulo Web de los agentes de SPADE, pasándole como parámetros la ruta de la carpeta que contiene todos los archivos estáticos. Esto permite utilizar todos los archivos CSS, JavaScript e iconos PNG, que fueron diseñados durante la primera versión, en el nuevo servidor.

Los middlewares y la configuración de la carpeta Locale para la implementación del sistema multilenguaje no tenían una fácil adaptación tras eliminar Django del proyecto, es por esto por lo que se ha decidido no implementarlo y dejarlo para futuras funcionalidades.

Estos son los cambios principales que se han realizado para adaptar la aplicación al nuevo agente launcher, que será el encargado de construir el mapa de navegación web, cargar los recursos estáticos y lanzar el servidor a través de la función *start* del módulo web. El agente launcher permitirá una comunicación con todos los agentes que participan durante la ejecución de una partida de pyGOMAS.

5.2 Funcionalidades

Una vez reconfigurado el proyecto, se pretende seguir con la implementación de las funcionalidades que habían quedado pendientes durante la primera versión y otras nuevas funcionalidades que permiten monitorizar e interactuar con los agentes durante el transcurso de una partida a tiempo real.

5.2.1. Lanzar agentes en cualquier servidor XMPP

Durante la primera versión se había implementado una funcionalidad que permitía ejecutar el manager, los agentes tropa y el render después de configurarlos. Debido a que este proyecto va dirigido a los estudiantes, donde cada uno de ellos utiliza su propia máquina, era necesario implementar un sistema donde se pueda lanzar agentes por separado en cualquier otro servidor XMPP. Actualmente solo se podía configurar el nombre del servidor XMPP en el manager, tras lanzarlo, los agentes tropa estaban directamente configurados para que sean lanzados en el mismo servidor, al igual que el render. Por este motivo, se ha implementado una nueva funcionalidad que permite añadir agentes a una partida, de una forma mucho más flexible y en cualquier servidor XMPP existente.

Para una mejor compresión de la utilidad de esta funcionalidad, pongamos el ejemplo de que nos encontremos en una sesión de prácticas donde se esté utilizando pyGOMAS. El profesor tendrá la posibilidad de lanzar únicamente el agente manager en un servidor XMPP ya existente. Una vez el agente manager este preparado, cada alumno podrá configurar un agente tropa en su propia máquina, y después lanzarlo en el mismo servidor XMPP donde había sido lanzado el manager, de manera que cada uno de los alumnos participará en la partida.

Para implementar esta funcionalidad, se ha añadido una nueva opción al menú “Unirse a partida”. Al hacer clic en esta opción se mostrará una interfaz de usuario donde se podrá seleccionar que agente se desea lanzar, puede ser el manager, una agente tropa, o el render. Una vez seleccionado el agente que queremos lanzar, se permitirá configurar el servidor XMPP donde queramos lanzar el agente. Para lanzar las tropas es importante añadir el nombre de un manager que ya esté conectado en cualquier servidor XMPP, al igual que el nombre del agente service, ya que no se pueden lanzar tropas en cualquier servidor XMPP donde no exista un agente manager.

Figura 16. Configuración y lanzamiento del agente manager en cualquier servidor XMPP

Figura 17. Configuración y lanzamiento del agente tropa en cualquier servidor XMPP

Figura 18. Configuración y lanzamiento del render en cualquier servidor XMPP

5.2.2. Monitorizar partida

Tras la implementación del nuevo agente launcher, se había abierto la posibilidad de que dicho agente pudiese comunicarse con otros que participan directamente en una partida. Esto permitía obtener datos a tiempo real sobre el estado actual de la partida. En este caso, esta nueva funcionalidad tiene como objetivo mostrar al usuario que agentes de cada equipo siguen con vida y también indicar aquellos que han muerto durante la partida.

Durante el punto 2.2 se ha comentado la herramienta que proporciona SPADE para implementar comportamientos de distintos tipos a los agentes y cómo es posible que se puedan intercambiar información a través de un sistema de mensajería. Poniendo en práctica esta herramienta, se ha desarrollado un nuevo comportamiento al agente launcher, que será el encargado de obtener una lista de todos los agentes tropa que siguen con vida durante la partida.

El agente service se encarga de registrar los servicios que ofrecen los agentes tropa. Este agente responde a peticiones de solicitud de servicios, y es conocedor de que unidades siguen vivas. En el archivo service.py se puede encontrar un comportamiento de tipo *CyclicBehaviour* llamado *GetServiceBehaviour* que espera a recibir mensajes de otros agentes. Una vez el agente recibe un mensaje, le devolverá al agente emisor una lista de todos los agentes tropa que siguen vivos. La construcción de esta lista dependerá del atributo “body” del mensaje. El cuerpo del mensaje está formado por la etiqueta “NAME”, que indica el tipo de tropa (medic, fieldop o soldier), aunque

también puede obtener como valor el nombre de un equipo (axis o allied). “TEAM” es la otra etiqueta del mensaje, que puede tener un valor que representa al equipo Eje, y otro valor que representa al equipo Aliado. De esta forma, el agente servicie puede construir un mensaje que indique los agentes tropa que siguen ofreciendo servicios y que, por lo tanto, siguen vivos.

Aprovechando el comportamiento ya definido del agente service, se ha implementado un nuevo comportamiento “getSoldiers”, de tipo *CyclicBehaviour* en el agente launcher. Este comportamiento primero envía un mensaje al agente service, que está constantemente esperando mensajes a través de su comportamiento “GetServiceBehaviour”, con las etiquetas NAME = “allied” y TEAM = “100”, tras enviar dicho mensaje, el agente launcher se queda esperando a recibir un mensaje del agente service con la lista de agentes tropa solicitada. El agente service, al recibir el mensaje, devuelve la lista que corresponde a los valores de las etiquetas, en este caso, devuelve todos los miembros del equipo Aliado. El valor de 100 en la etiqueta TEAM corresponde al equipo Aliado, esto es interpretado por el agente service a través de una ontología ya diseñada en pyGOMAS. Una vez el agente launcher recibe el mensaje del agente service, cuyo cuerpo contiene una la lista de los agentes vivos del equipo Aliado, se construye un nuevo mensaje, esta vez con las etiquetas NAME = “axis” y TEAM = “200”. Al igual que el mensaje anterior, el agente service construye una nueva lista que contiene todos los agentes pertenecientes al equipo Eje que siguen vivos y la envía al launcher. Este comportamiento del launcher es cíclico, lo que significa que, una vez ejecutado, estará constantemente consultando y obteniendo las listas de los agentes tropa de cada equipo, por lo que, si durante la partida alguna unidad es eliminada, en la lista obtenida no aparecerá dicha unidad.

Una vez implementado el intercambio de mensajes entre el launcher y el service, se diseña una interfaz para que el usuario sea capaz de visualizar que tropas de cada equipo están participando en la partida, y cuáles de ellas han sido eliminadas. Para ello se diseña una nueva vista, la cual contiene dos tablas, una para cada equipo, donde aparecen los nombres de los agentes tropa que están participando en la partida. Cada agente tiene un icono que indica su estado.

Para poder acceder a esta lista, se crea una nueva opción en el menú principal de la aplicación “Monitorizar partida”. Cuando el usuario seleccione esta opción, se abrirá un pequeño formulario donde indicará el servidor XMPP donde se ha lanzado el agente service, y el nombre (JID) de dicho agente.

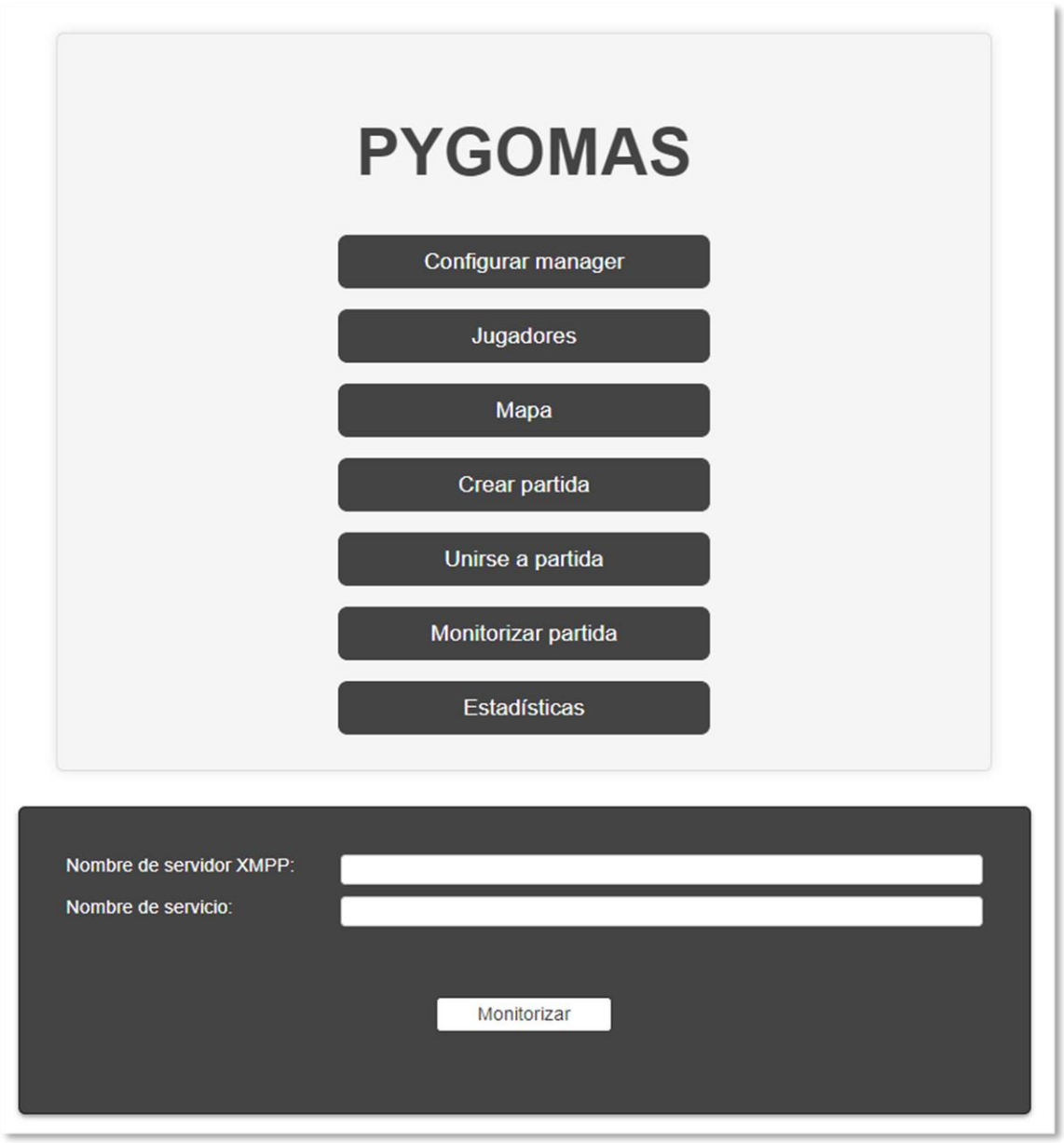


Figura 19. Monitorizar partida

Cuando el usuario complete estos dos campos, el agente launcher ejecuta el comportamiento "GetSoldiers", y construye el nombre completo del agente service al que tiene que enviar el mensaje. El nombre del agente Service tiene la estructura {JID}@{XMPPServer}, por este motivo se pide al usuario estos dos campos. Es importante destacar que, para acceder a la monitorización de una partida, la partida debe estar en curso, ya que, si el agente service no existe, ningún agente podrá recibir los mensajes que está tratando de enviar el agente launcher. Una vez

el usuario haga clic en el botón “Monitorizar” tras rellenar los campos, podrá visualizar la siguiente vista:

Figura 20. Monitorizar partida

Figura 21. Monitorizar partida (agentes muertos)

En las tablas se puede observar que unidas están vivas y cuales no a través de los iconos, estos iconos se actualizan a tiempo real a medida que los estados de los agentes se modifican. Sin embargo, para conseguir este comportamiento se resolvieron un par de problemas que se presentaron durante el desarrollo.

El primero de ellos es que el agente service solamente nos indica las unidades que siguen con vida, esto significa que, si obtenemos una lista de un equipo con 10 unidades y, tras pasar un tiempo, mueren 4 unidades de ese mismo equipo, la lista estaría formada por solamente 6 unidades. Por esta razón, no es suficiente con obtener las listas y mostrarlas directamente. Se ha diseñado lógica mediante JavaScript, que continuamente está comparando una lista inicial con las listas que el launcher está obteniendo de forma cíclica por parte del service, identificando que unidades se han eliminado de las listas. En caso de que una unidad existente en la lista inicial ya no se encuentre en una nueva lista, significara que la unidad a muerto.

El otro problema que surgió es la actualización de la vista y los datos en tiempo real. Para poder actualizar el estado de las tropas era necesario recargar la página, lo cual se consideró algo muy poco práctico. Para solucionar este problema se utiliza la tecnología AJAX, que permite actualizar partes específicas de una página web sin necesidad de recargar toda la página. AJAX permite realizar solicitudes al servidor y recibir respuestas en formato de datos de forma asíncrona. Mediante AJAX se realizan peticiones de forma iterativa a un nuevo controlador desarrollado, “getAgents”, mostrado en la figura 15, que corresponde con el mapa de navegación. Al implementar esto, el controlador que corresponde con la vista “monitorizar” solamente obtiene la lista de agentes vivos iniciales, una vez procesada esta petición, AJAX se encarga de realizar peticiones al controlador “getAgents” en segundo plano, de forma que no es necesario actualizar la página constantemente para que se modifiquen los estados, obteniendo así una actualización a tiempo real a medida que avanza la partida.

5.2.3. Ver estadísticas

Tras finalizar una partida, pyGOMAS genera un archivo .txt donde se pueden visualizar distintas estadísticas de cada equipo como las tropas supervivientes, los disparos recibidos, los disparos dados a tropas enemigas, el porcentaje de vida restante, los paquetes médicos y de munición generados y una gran cantidad de datos. El objetivo consiste en que los usuarios puedan visualizar de forma cómoda estas estadísticas mediante una interfaz, además de que cada una de las partidas creadas en la carpeta “Saves” mencionada en el apartado 4.2.2 contenga el archivo .txt con las estadísticas de la última partida ejecutada.

Anteriormente, tras finalizar cualquier partida, el archivo de estadísticas se generaba en el directorio raíz de pyGOMAS. Cuando se finalizaba una segunda partida, el nuevo archivo generado sobrescribía al antiguo, de forma que solamente podía existir un único archivo. La idea es que, tras finalizar cualquier partida, el archivo quede almacenado en la carpeta de su partida correspondiente, por ejemplo, si ejecutamos una partida con el guardado “Game1”, tras finalizar el archivo quedará guardado en esa carpeta, posteriormente se podrá ejecutar una nueva partida con el guardado ‘Game2’, que también guardará sus estadísticas, permitiendo almacenar distintos archivos de distintas partidas.

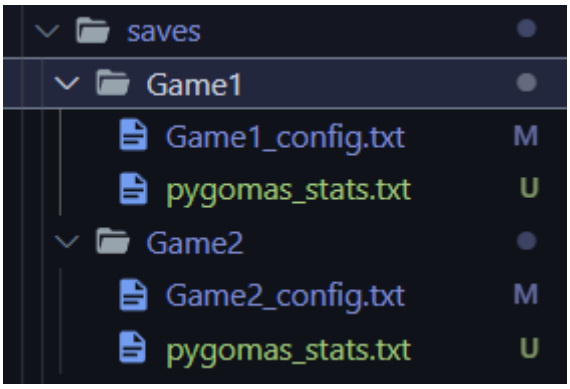
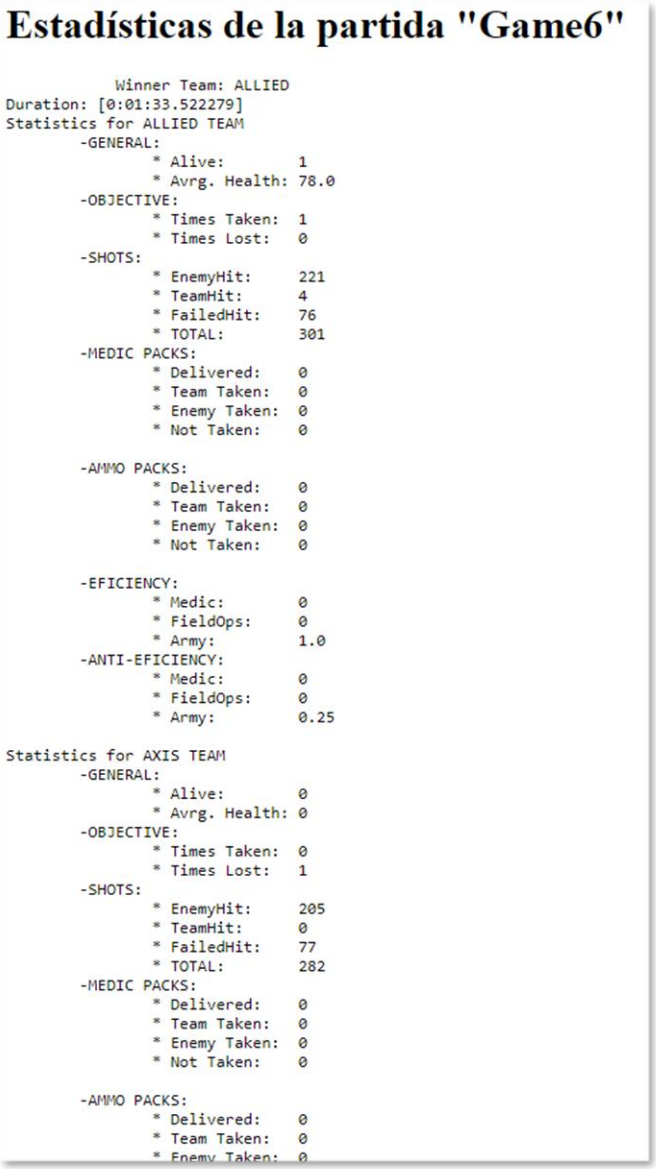


Figura 22. Ejemplo de directorio de las partidas guardadas

Para poder lograr esto, es necesario que el agente launcher sea conocedor del momento en el que finaliza una partida, para poder mover el archivo a la carpeta de guardado desde donde ha sido lanzada. Para ello, se ha desarrollado un nuevo comportamiento en el agente launcher llamado "InformFinishedGame", cuya función es esperar un mensaje de confirmación de parte del agente Manager, indicando que la partida en curso ha sido finalizada. Una vez se reciba el mensaje de confirmación, el archivo de estadísticas se moverá a la carpeta de guardado que corresponde con la partida de guardado en la que el usuario se encuentra.

El agente manager también ha sido modificado para que envíe el mensaje de confirmación de finalización de la partida al agente launcher. Para ello se ha aprovechado la función ya implementada "inform_game_finished", la cual se encarga de informar a todos los agentes de que una partida ha finalizado. Esta función se ha ampliado para que también envíe un mensaje al agente launcher. Además, esta función se ejecuta cuando la partida finaliza de cualquiera forma, ya sea por la victoria del equipo Eje, la del equipo Aliado o por la expiración del tiempo de partida.

Para que el usuario pueda ver las estadísticas de la última partida finalizada, se ha añadido una nueva opción en el menú principal de le aplicación llamada "Estadísticas", la cual redirigirá al usuario a una nueva vista que contendrá todo el contenido del archivo .txt correspondiente a la partida donde está situado. También se ha añadido esta opción en la vista "monitorizar" mencionada en el apartado 5.2.2, la cual también redirige al usuario a la nueva vista para ver las estadísticas.



Estadísticas de la partida "Game6"

```
           Winner Team: ALLIED
Duration: [0:01:33.522279]
Statistics for ALLIED TEAM
        -GENERAL:
                * Alive:        1
                * Avrg. Health: 78.0
        -OBJECTIVE:
                * Times Taken:  1
                * Times Lost:   0
        -SHOTS:
                * EnemyHit:     221
                * TeamHit:      4
                * FailedHit:    76
                * TOTAL:        301
        -MEDIC PACKS:
                * Delivered:    0
                * Team Taken:   0
                * Enemy Taken:  0
                * Not Taken:    0

        -AMMO PACKS:
                * Delivered:    0
                * Team Taken:   0
                * Enemy Taken:  0
                * Not Taken:    0

        -EFICIENCY:
                * Medic:        0
                * FieldOps:     0
                * Army:         1.0
        -ANTI-EFICIENCY:
                * Medic:        0
                * FieldOps:     0
                * Army:         0.25

Statistics for AXIS TEAM
        -GENERAL:
                * Alive:        0
                * Avrg. Health: 0
        -OBJECTIVE:
                * Times Taken:  0
                * Times Lost:   1
        -SHOTS:
                * EnemyHit:     205
                * TeamHit:      0
                * FailedHit:    77
                * TOTAL:        282
        -MEDIC PACKS:
                * Delivered:    0
                * Team Taken:   0
                * Enemy Taken:  0
                * Not Taken:    0

        -AMMO PACKS:
                * Delivered:    0
                * Team Taken:   0
                * Enemy Taken:  0
```

Figura 23. Estadísticas de la última partida ejecutada en "Game6".

5.2.4. Cerrar Manager

Para lanzar una nueva partida, el primer paso es ejecutar el agente manager. A partir de este punto, nuevos agentes se van uniendo al servidor XMPP, como el agente service y también los agentes tropa, qué, a su vez, generan paquetes médicos y paquetes de munición, que también son agentes que forman parte de la partida y están conectados al servidor. A continuación, se muestra una imagen de todos los agentes pueden llegar a estar conectados durante una ejecución de pyGOMAS.

1	allied_field_0	9r5tv0xtrn	Autenticado	Conectado
2	allied_medic_0	4x1hhckn4c	Autenticado	Conectado
3	allied_soldier_0	4qfb6yjyzx	Autenticado	Conectado
4	ammopack_axis_field_0_1	6fnrypkcp9	Autenticado	Conectado
5	ammopack_axis_field_0_2	guovz8xa9	Autenticado	Conectado
6	ammopack_axis_field_0_3	29mkozw710	Autenticado	Conectado
7	ammopack_axis_field_0_4	42dnyxu32z	Autenticado	Conectado
8	axis_field_0	6zjoarjv23	Autenticado	Conectado
9	axis_medic_0	1xahy1a7fj	Autenticado	Conectado
10	axis_soldier_0	4s6gfiozs0	Autenticado	Conectado
11	launcher	5wv6kjqbm3	Autenticado	Conectado
12	manager	6nyb61pnbs	Autenticado	Conectado
13	objectivepack	62ob5tbfkh	Autenticado	Conectado
14	service	25r3l9rw75	Autenticado	Conectado

Figura 24. Agentes conectados durante el transcurso de una partida de pyGOMAS

Tras finalizar la partida, el agente Manager informa al resto de agentes conectados al servidor, a través de la función “inform_game_finished”, que la partida ha llegado a su fin. Cuando los agentes reciben esta información a través de mensajes, se detienen, desconectándose del servidor. Este proceso solamente sucede cuando uno de los dos equipos sale ganador, o por la expiración del tiempo de partida. En caso de que el agente manager se desconecta de forma manual, los otros agentes no finalizarán, y esto puedo ocasionar problemas si se vuelve a lanzar otra partida en la que los agentes contengan el mismo nombre.

Se pretende implementar una funcionalidad que permita al usuario detener una partida durante su ejecución. Esto permitirá realizar pruebas de ejecución de partidas con distintos parámetros en su configuración sin la necesidad de esperar a que finalicen, o simplemente, detener una partida en caso de que se haya ejecutado por error.

Para añadir esta nueva funcionalidad, se ha diseñado un nuevo comportamiento para el agente manager, llamado “killManager. Este comportamiento, de tipo ‘*OneShotBehaviour*, se inicia en el instante en el que se crea el agente manager, es decir, en su función “setup”. El funcionamiento de este nuevo comportamiento consiste en esperar un mensaje cuyo cuerpo contenga la cadena “kill”, una vez recibido, se lanza la función mencionada anteriormente “inform_game_finished”, que detendrá al resto de los agentes y al propio agente manager.

También es necesario añadir un nuevo comportamiento que se encargue de enviar el mensaje “kill” directamente al agente manager, el agente encargado de realizar esto es el launcher, al que se le ha diseñado un nuevo comportamiento llamado “killManager”. Este comportamiento se inicia cuando el usuario hace clic sobre el botón “Detener partida” en la vista de monitorización:

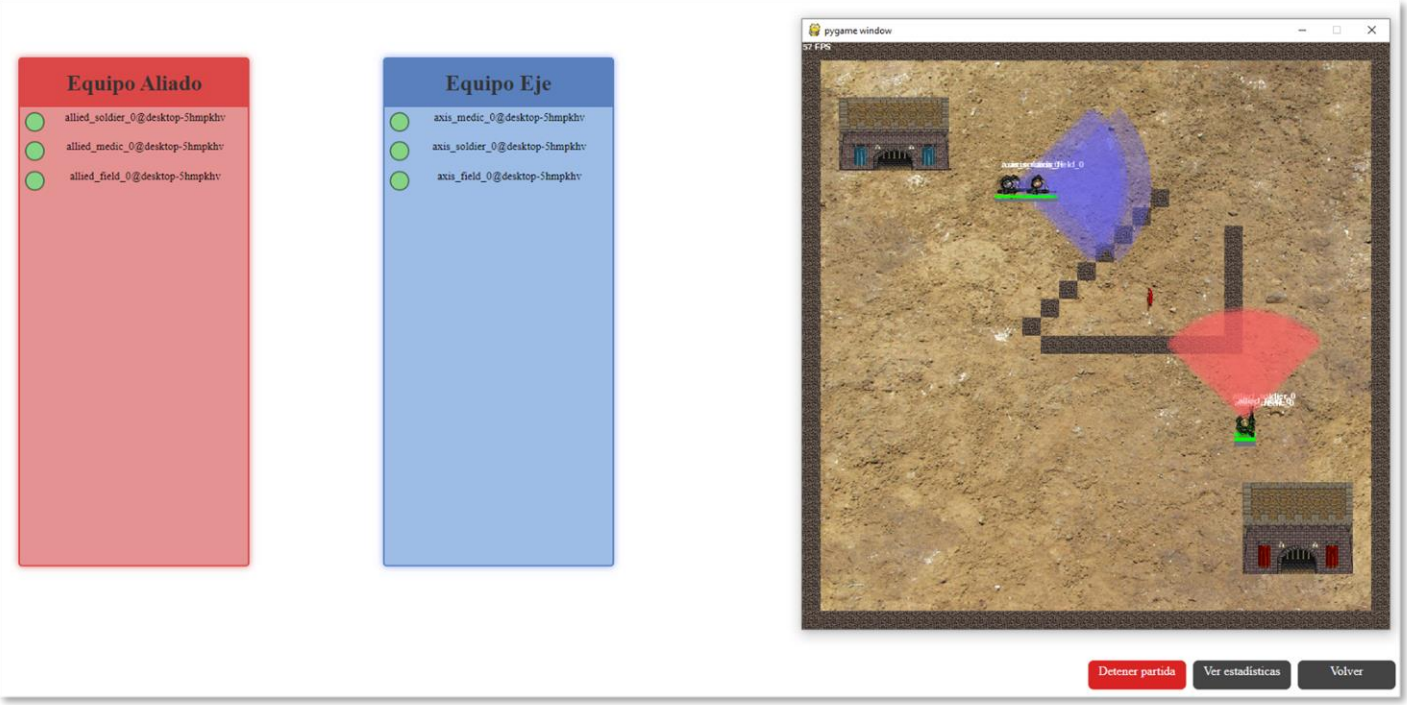


Figura 25. Vista para la monitorización de una partida de pyGOMAS

Al hacer clicc en “Detener partida”, se realiza una petición al servidor. La petición es manejada por el controlador, que se encarga de asignar el nombre del manager y el nombre del servidor a los atributos del nuevo comportamiento “killManager” a través del archivo de configuración de cada guardado de la carpeta “Saves” y posteriormente, ejecutar el comportamiento. “killManager” se encarga de construir el nombre completo del agente Manager, que tiene el formato {nombre_manager}@{servidorXMPP}, y enviarle el mensaje correspondiente para que este se detenga de forma limpia, es decir, deteniendo todos los agentes que participan durante la partida.

5.2.5. Cerrar Launcher

La posibilidad de que el usuario puede cerrar el agente launcher en cualquier momento era de vital importancia, ya que, en el caso de que se cerrara el navegador donde la aplicación se estaba ejecutando, el launcher seguía conectado al servidor XMPP. Esto podía ocasionar problemas debido a la posibilidad de lanzar distintos agentes launcher, con el mismo nombre, al mismo servidor XMPP. Teniendo en cuenta que el agente launcher establece comunicación con otros agentes que forman parte de pyGOMAS, como el Manager o el service, se podían originar ciertos conflictos en el momento de recibir mensajes en caso de que existiesen dos agentes con el mismo nombre. Por este motivo, se ha implementado esta funcionalidad, que permite al usuario apagar directamente el launcher.

Para lograr esto, se ha implementado una nueva ruta en la aplicación “/pygomas/exit”, que no devuelve ninguna vista y cuya petición es manejada por una función controladora. En la función controladora se realiza el proceso de detención del agente a través de la función *stop*, proporcionada por SPADE.

Para que el usuario pueda realizar la petición, se ha incluido en la interfaz un botón que envía al usuario directamente a la URL “pygomas/exit” mencionada anteriormente y de esta forma, el launcher se detiene y queda desconectado del servidor XMPP.

Figura 26. Menú principal con la opción “Cerrar launcher” incluida

6. Pruebas

En esta sección se sigue todo el progreso de dos ejecuciones completas que un alumno puede llegar a emplear durante una de sus prácticas, para ello, las pruebas se centraran el ejecutar una partida con una cierta configuración, poder visualizarla, monitorizarla y posteriormente ver los resultados.

6.1. Prueba en local

Primero se ha creado una nueva partida llamada “Prueba_Local”, lo que genera una nueva subcarpeta en la carpeta “Saves” con su archivo de configuración correspondiente.

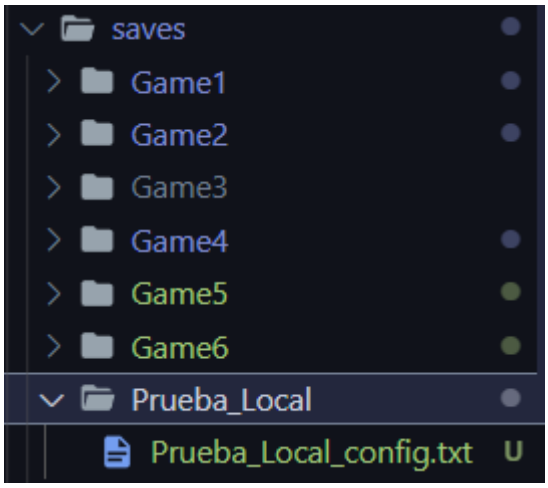


Figura 27. Creación de nueva partida “Prueba_Local”

A continuación, se han modificado todos los parámetros del agente manager mediante el apartado del menú principal “Configurar Manager” con los siguientes valores:

Figura 28. Nueva configuración del manager para la partida "Prueba_Local"

También se ha creado un nuevo archivo JSON donde se estable la configuración de todos los agentes tropa que participarán en la partida a través del apartado "Jugadores" en el menú principal.

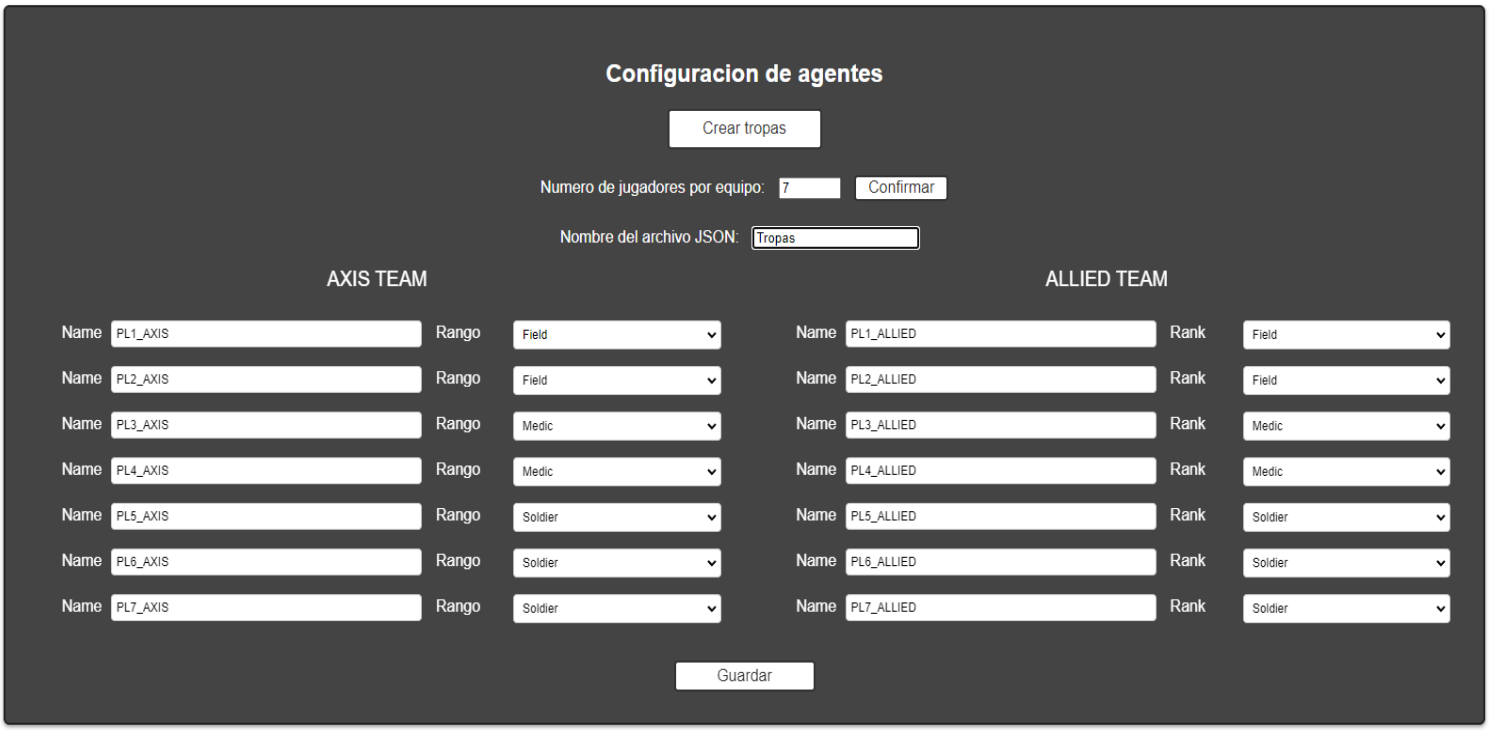


Figura 29. Configuración de los agentes tropa

Tras configurar todos los agentes tropa (nombre y tipo), se ha generado un archivo JSON llamado "Tropas.json", que contiene dicha configuración además de otros valores que han sido obtenidos directamente de la configuración del manager.

```
{
    "host": "desktop-5hmpkhv",
    "manager": "managerLocal",
    "service": "serviceLocal",
    "axis": [
        {
            "rank": "BDIFieldOp",
            "name": "PL1_AXIS",
            "password": "secret"
        },
        {
            "rank": "BDIFieldOp",
            "name": "PL2_AXIS",
            "password": "secret"
        },
        {
            "rank": "BDIMedic",
            "name": "PL3_AXIS",
            "password": "secret"
        },
```


Figura 30. Fragmento de Tropas.json

Se ha creado un nuevo mapa, a través de la opción “Mapa” en el menú principal.

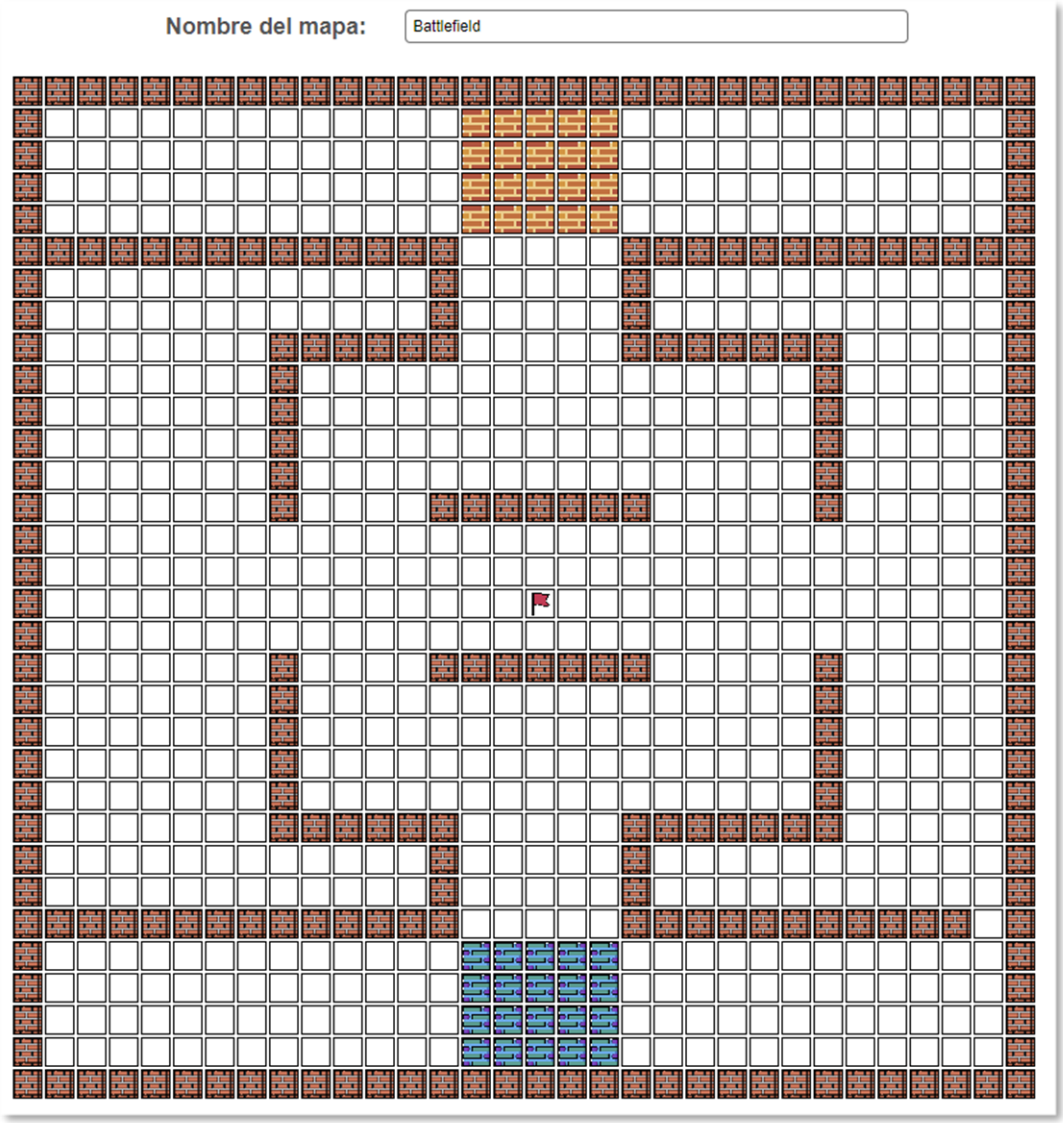


Figura 31. Nuevo mapa “Battlefield”

Tras diseñar, guardar y asignar el mapa, utilizando todas las funciones que proporciona el editor de mapas, se generan los archivos correspondientes que pyGOMAS utiliza para simular el campo de batalla. Estos dos archivos representan la estructura del mapa y donde se sitúan las parades y las coordenadas de los elementos como las zonas de aparición y el objetivo o bandera. Los archivos .txt se generan dentro de la carpeta Battlefield, también autogenerada en la carpeta "Maps".

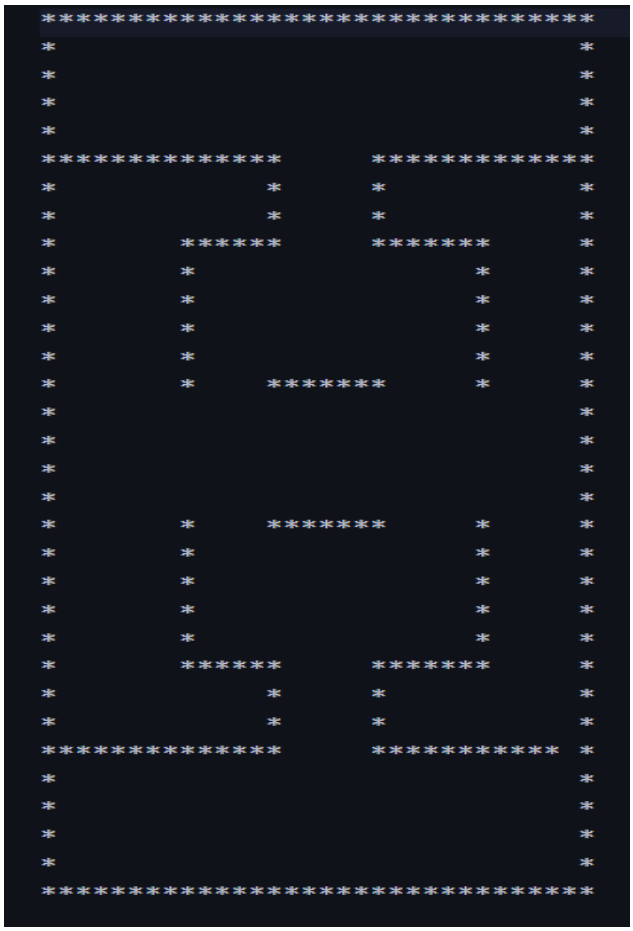


Figura 32. Contenido del archivo "Battlefield_cost.txt"

```
[pGomas]
pGomas_OBJECTIVE: 16 16
pGomas_SPAWN_ALLIED: 14 1 18 4
pGomas_SPAWN_AXIS: 14 27 18 30
pGomas_COST_MAP: 32 32 Battlefield_cost.txt
[pGomas]
```

Figura 33. Contenido del archivo "Battlefield.txt"

Una vez configurados tanto el Manager, como las tropas y el diseño del nuevo mapa, se pueden observar los cambios en la configuración de la partida "Prueba_Local".

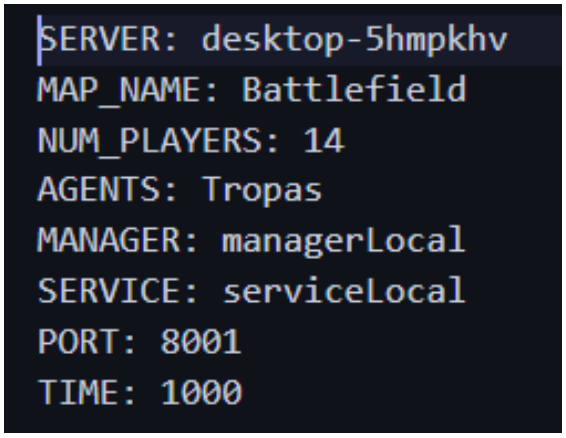


Figura 34. Configuración personalizada de "Prueba_Local"

A continuación, se procede al lanzamiento de la partida con los parámetros de configuración vistos anteriormente mediante la opción "Crear partida" en el menú principal.

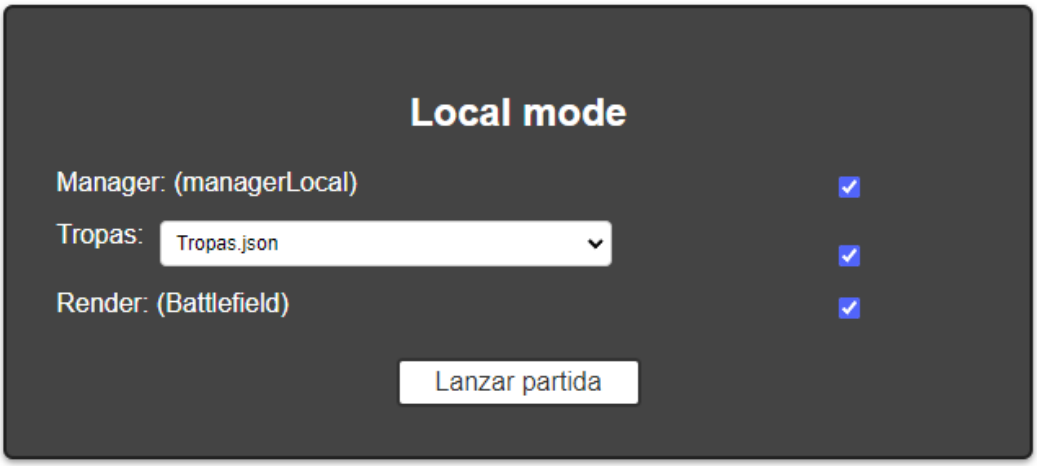


Figura 35. Lanzamiento de una partida pyGOMAS

En este caso se ha decidido lanzar el agente manager, que es imprescindible, los agentes tropa y el render, este último permite visualizar la partida a tiempo real. Tras darle al botón "Lanzar partida" y esperar unos pocos segundos para que el agente manager este preparado y se lancen las tropas, se abrirá el render, donde será posible visualizar la partida.

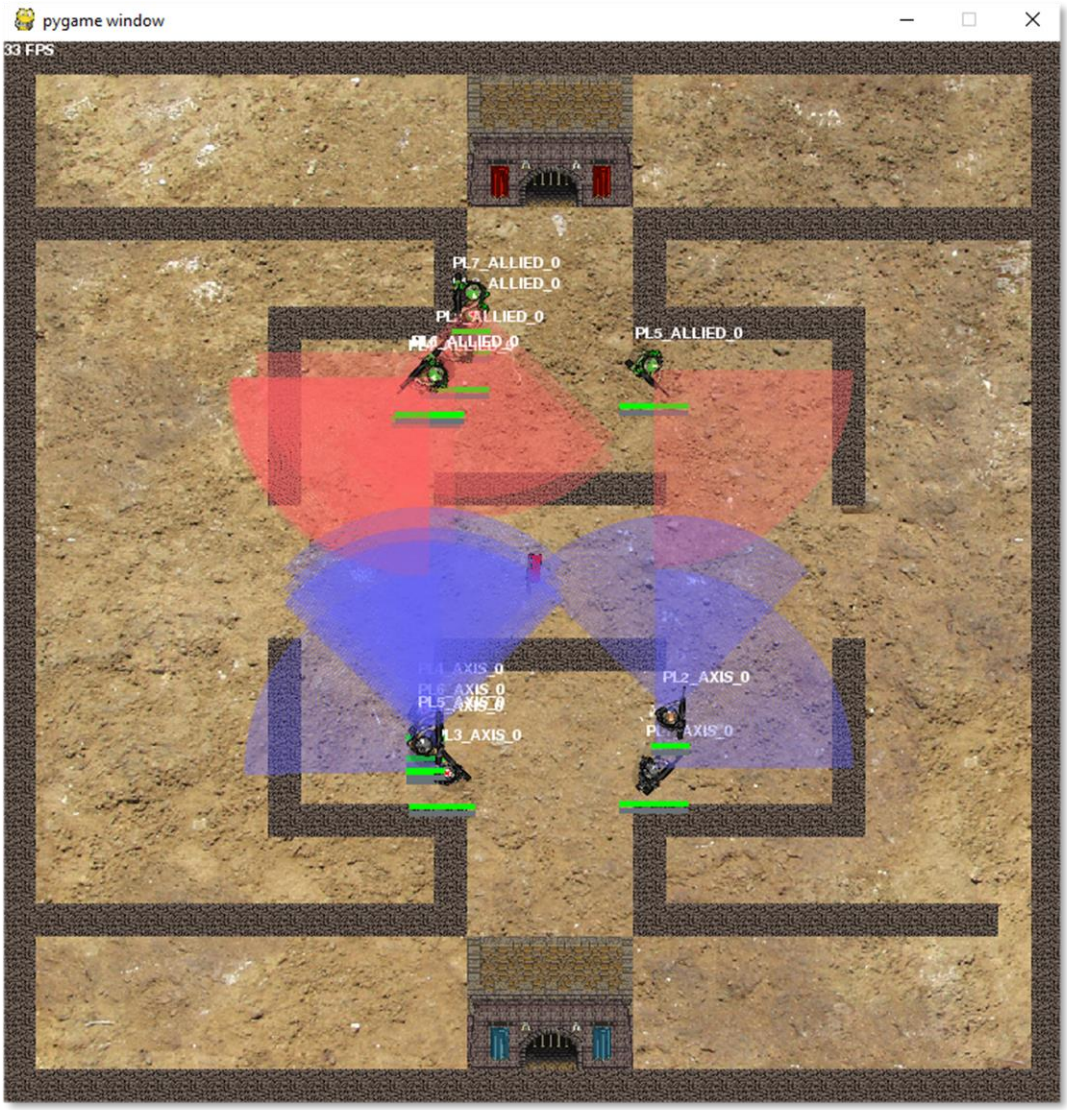
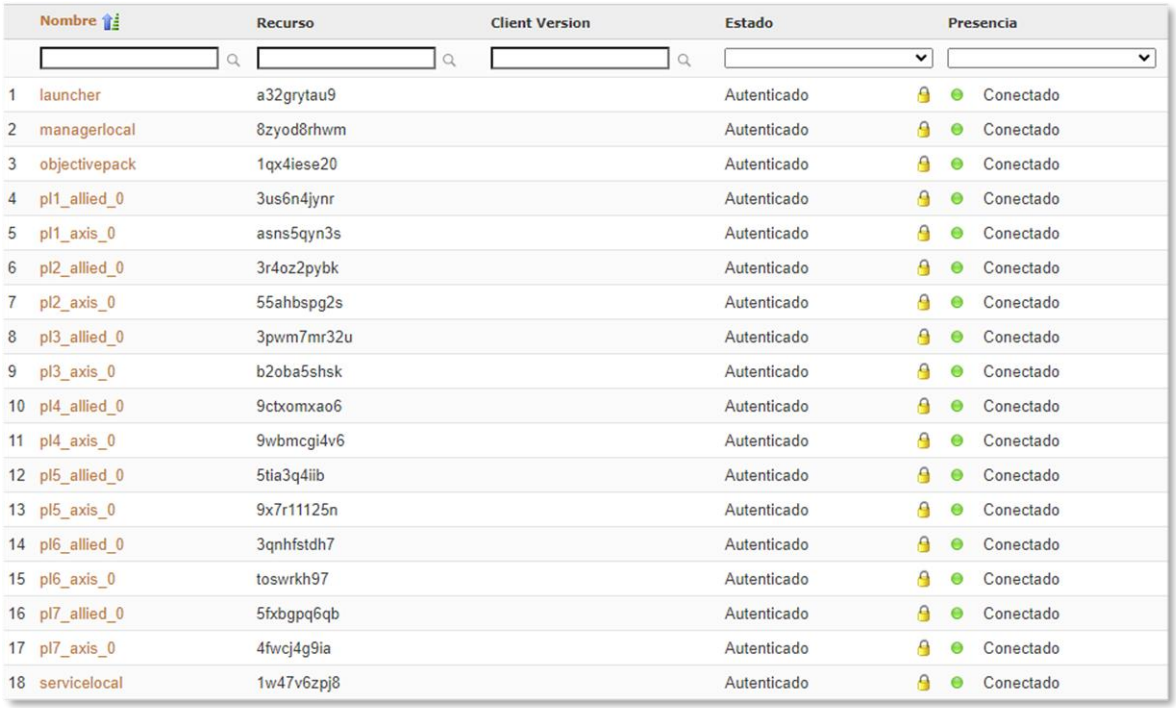


Figura 36. Partida en curso con la configuración personalizada



	Nombre	Recurso	Client Version	Estado	Presencia
1	launcher	a32grytau9		Autenticado	Conectado
2	managerlocal	8zyod8rhwm		Autenticado	Conectado
3	objectivepack	1qx4iese20		Autenticado	Conectado
4	pl1_allied_0	3us6n4jynr		Autenticado	Conectado
5	pl1_axis_0	asns5qyn3s		Autenticado	Conectado
6	pl2_allied_0	3r4oz2pybk		Autenticado	Conectado
7	pl2_axis_0	55ahbspg2s		Autenticado	Conectado
8	pl3_allied_0	3pwm7mr32u		Autenticado	Conectado
9	pl3_axis_0	b2oba5shsk		Autenticado	Conectado
10	pl4_allied_0	9ctxomxao6		Autenticado	Conectado
11	pl4_axis_0	9wbmcgi4v6		Autenticado	Conectado
12	pl5_allied_0	5tia3q4iib		Autenticado	Conectado
13	pl5_axis_0	9x7r11125n		Autenticado	Conectado
14	pl6_allied_0	3qnhfstdh7		Autenticado	Conectado
15	pl6_axis_0	toswrkh97		Autenticado	Conectado
16	pl7_allied_0	5fxbgpq6qb		Autenticado	Conectado
17	pl7_axis_0	4fwcj4g9ia		Autenticado	Conectado
18	servicelocal	1w47v6zpj8		Autenticado	Conectado

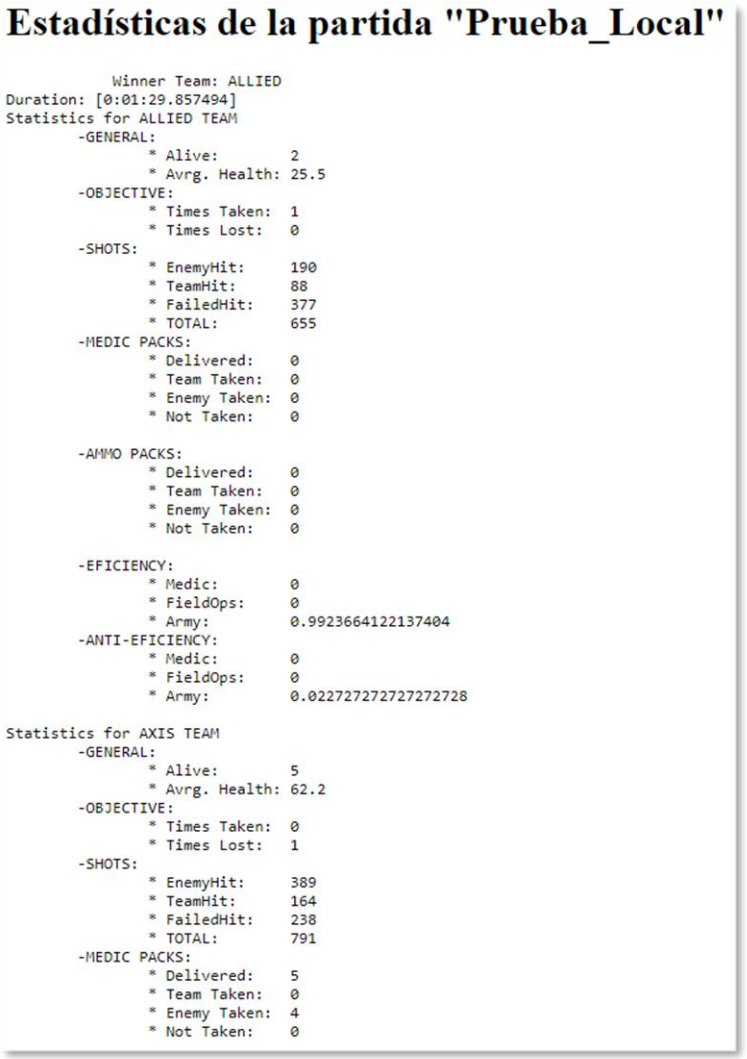
Figura 37. Agentes SPADE conectados durante el transcurso de la partida

A partir de este momento, se abre la posibilidad de monitorizar la partida en curso. Esta opción está disponible en la opción “Monitorizar” del menú principal, donde se ha introducido el nombre del servidor XMPP y el JID del agente service conectado al servidor. Las dos imágenes siguientes muestran el estado de los agentes tropa, la primera imagen corresponde al inicio de la partida, donde todas las tropas siguen con vida, en la segunda algunas tropas tienen el icono de una esquela, lo que significa que ya han sido eliminados.

Figura 38. Estado de los agentes durante el inicio de la partida

Figura 39. Estado de los agentes con la partida avanzada

Cuando la partida llega a su fin y hay un ganador, se abre la posibilidad de ver las estadísticas desde la vista de monitorización o desde el menú principal.



Estadísticas de la partida "Prueba_Local"

```
           Winner Team: ALLIED
Duration: [0:01:29.857494]
Statistics for ALLIED TEAM
        -GENERAL:
                * Alive:        2
                * Avrg. Health: 25.5
        -OBJECTIVE:
                * Times Taken:  1
                * Times Lost:   0
        -SHOTS:
                * EnemyHit:     190
                * TeamHit:      88
                * FailedHit:    377
                * TOTAL:        655
        -MEDIC PACKS:
                * Delivered:    0
                * Team Taken:   0
                * Enemy Taken:  0
                * Not Taken:    0

        -AMMO PACKS:
                * Delivered:    0
                * Team Taken:   0
                * Enemy Taken:  0
                * Not Taken:    0

        -EFICIENCY:
                * Medic:        0
                * FieldOps:     0
                * Army:         0.9923664122137404
        -ANTI-EFICIENCY:
                * Medic:        0
                * FieldOps:     0
                * Army:         0.022727272727272728

Statistics for AXIS TEAM
        -GENERAL:
                * Alive:        5
                * Avrg. Health: 62.2
        -OBJECTIVE:
                * Times Taken:  0
                * Times Lost:   1
        -SHOTS:
                * EnemyHit:     389
                * TeamHit:      164
                * FailedHit:    238
                * TOTAL:        791
        -MEDIC PACKS:
                * Delivered:    5
                * Team Taken:   0
                * Enemy Taken:  4
                * Not Taken:    0
```

Figura 40. Fragmento de las estadísticas de la última ejecución de pygomas

Con esta ejecución se abarcan las principales herramientas que proporciona la aplicación cuando se lanza una partida en local.

6.1. Prueba con distintas máquinas.

La siguiente prueba será muy similar a la anterior, con la diferencia de que los agentes serán lanzados desde distintas máquinas para que se conecten al mismo servidor XMPP. Esto puede servir como una simulación de lo que pueden llegar a realizar los estudiantes durante el transcurso de una práctica, la idea principal es que cada estudiante ejecute en el servidor su propio agente tropa.

Primero se ha iniciado la aplicación desde una primera máquina. Una vez abierta se ha creado una nueva partida llamada “Test_Maquina_A”. En el menú principal se accede a la opción “Unirse a partida”, donde se selecciona que tipo de agente queremos ejecutar. En este caso, la maquina “A” lanzará el agente manager.

Figura 41. Lanzamiento del agente manager desde la máquina A.

Se ha utilizado el mapa creado en la anterior prueba llamado “Battlefield”, además se ha especificado que participaran 4 jugadores en esta partida, esto significa que una vez lanzado el manager se quedará esperando hasta que se lancen 4 agentes tropas desde cualquier otra máquina.

Tras lanzar el manager desde “A”, se ha accedido a la aplicación desde otra máquina que llamaremos “B”, y, al igual que antes, se ha seleccionado la opción “Unirse a partida” del menú principal. Tras rellenar la configuración en el apartado “Tropas”, visible en la Figura 41, se procede a lanzar cada uno de los agentes tropa. En este caso, el último proceso se ha realizado 4 veces, ya que son un total de 4 jugadores los que solicita el agente manager. La idea es que cada estudiante ejecute un agente diferente, en este caso no es posible replicar dicho comportamiento ya que solo se dispone de dos máquinas, por este motivo todos los agentes tropa se han lanzado desde la máquina B.

Figura 42. Lanzamiento de la tropa “PLAYER1” desde la máquina B.

Tras lanzar 4 soldados, se ha monitorizado la partida desde ambas máquinas, donde también existe la opción de ver las estadísticas completas una vez finalizada la partida.

Figura 43.Monitorización desde la máquina A

7. Conclusiones

A lo largo de este apartado se desarrollará una conclusión enfocada en el cumplimiento de los objetivos iniciales, el aprendizaje adquirido personalmente y una serie de implementaciones no realizadas que se podrían añadir en versiones futuras.

7.1 Cumplimiento de objetivos

Uno de los aspectos más destacados de este proyecto ha sido el cumplimiento efectivo de los objetivos establecidos desde el inicio. Se considera que los principales objetivos propuestos en el apartado 1.2 han sido cumplidos con éxito, ya que, se ha realizado un análisis de las necesidades de los estudiantes y se ha realizado una implementación de una aplicación que sirve como herramienta para la configuración, la ejecución y monitorización de las partidas de pyGOMAS. Además, también se han identificado funcionalidades que pueden ser añadidas en versiones futuras, y que se comentarán más adelante.

7.2 Aprendizaje y desarrollo

Durante el desarrollo del proyecto, han ido surgiendo distintos problemas y obstáculos a superar, esto es debido, en gran parte, a la poca experiencia con las tecnologías usadas. No obstante, los contratiempos se han ido superando y han proporcionado una oportunidad excepcional para el aprendizaje y desarrollo de habilidades en distintas áreas.

En primer lugar, se ha adquirido un conocimiento profundo en la implementación de agentes inteligentes utilizando SPADE, lo que ha ampliado las capacidades en el campo de los agentes inteligentes como concepto general y la programación asíncrona, que está muy relacionada en este tipo de campo.

Otro punto que destacar es el lenguaje de programación utilizado, Python. La elección de Python como lenguaje principal del desarrollo ha demostrado ser acertada y ha aportado numerosas ventajas a lo largo del proceso, además de proporcionar conocimientos propios de un lenguaje de programación tan activo como lo es Python en la actualidad.

A pesar de que Django es eliminado del proyecto en la segunda versión, también se han adquirido conocimientos muy valiosos sobre el desarrollo web utilizando este potente framework tan conocido en la actualidad. Aunque Django no tenga ningún impacto en la versión final del proyecto, la adquisición de conocimientos puede ser muy útil para el desarrollo de aplicaciones web futuras.

7.3 Trabajos futuros

Se pueden desarrollar más versiones de pyGOMAS y en concreto añadir nuevas funcionalidades. En este apartado se mencionan algunas de las funcionalidades que fueron valoradas y al final no fueron añadidas en esta versión, dejando la oportunidad para sean añadidas en versiones futuras.

- **Enviar órdenes a las tropas**. Esta funcionalidad consiste en enviar ordenes especificas a las tropas durante el transcurso de una partida como moverse, disparar, morir, crear paquetes de medicina o crear paquetes de munición.
- **Implementar distintos modos de juego**. Consiste en ofrecer al usuario la opción de seleccionar un modo de juego distinto al principal, es decir, un equipo contra otro. El modo adicional que se pensó inicialmente fue el "Todos contra todos", donde desaparecen los equipos y cada uno de los agentes actúa de forma individual.
- **Crear mapas con distintos tamaños**. En la versión actual, el tamaño del mapa es fijo, ya que corresponde a un mapa de tamaño 32x32 y esto no se puede modificar. Inicialmente se pensó en implementar la creación de mapas con distintos tamaños, ya sean más grandes o pequeños, añadiendo algún tipo de límite.
- **Sistema multilenguaje**. Volver a implementar un sistema multilenguaje, como el que se había implementado durante la primera versión del proyecto.
-

Se considera que estos tres puntos mencionados anteriormente, podrían ser implementaciones muy interesantes que enriquecerían el proyecto y que podrían ser añadidas en versiones futuras.

7.4 Relación del trabajo con los estudios cursados.

En este apartado se comenta la relación entre la formación académica y el proyecto. Se nombrarán algunas de las asignaturas cursadas que se considera que han tenido mayor peso y han aportado más conocimientos para la realización del trabajo.

- Sistemas inteligentes. Esta asignatura se aportaron conceptos básicos de las IA basadas en reglas.
- Proyecto de ingeniería del Software. Se obtuvo formación de cómo abordar un proyecto software, desde la planificación, hasta las pruebas.
- Ingeniería software. En esta asignatura, se estudió un proyecto software ya implementado, su estudió su arquitectura y el análisis de requisitos.
- Concurrencia y sistemas distribuidos. Se estudió el concepto de la programación asíncrona y como se pueden ejecutar varias tareas al mismo tiempo.
- Interfaces persona computador. Se estudiaron conceptos para el diseño de interfaces efectivas y amigables para los usuarios.
- Mostrar más información durante la monitorización. La idea sería mostras algunos más datos relacionados con los agentes tropa durante la tarea de monitorización. Estos datos podrían ser la vida y la munición.

8. Bibliografía

[1] L. Rouhiainen, «Inteligencia artificial,» Alienta Editorial, Madrid, 2018.

[2] V. J. Julián y V. J. Botti, «Estudio de métodos de desarrollo de sistemas multiagente,» *Inteligencia Artificial. Revista Iberoamericana,* vol. 7, nº 18, pp. 65-80, 2003.

[3] G. Andrighetto, G. Governatori y P. Noriega, Normative Multi-Agent, 2013.

[4] S. Kumar y U. Kumar, «Java agent development framework.,» *International Journal Research,* vol. 1, nº 9, pp. 1002-1025, 2014.

[5] F. Chiacchio, «Agent-based modeling of the immune system: NetLogo,» *BioMed research international,* vol. 2014, 2014.

[6] S. Luck, G. C. Balan y L. Panait, «MASON: A Java Multi-Agent Simulation Library,» *Proceedings of Agent 2003 Conference on Challenges in Social Simulation,* vol. 9, nº 9, 2003.

[7] D. Muravev, H. Hu, A. Rakhmangulov y P. Mishkurov, «Multi-agent optimization of the intermodal terminal main parameters by using AnyLogic simulation platform: Case study on the Ningbo-Zhoushan Port,» *International Journal of Information Managemen,* vol. 57, pp. 102 - 133, 2021.

[8] J. Palanca, «Spade 3.3.0 documentation,» 2020. [En línea]. Available: https://spade-mas.readthedocs.io/en/latest/index.html.

[9] J. Palanca, «SPADE 3: Supporting the New Generation,» *IEEE Access,* vol. 8, p. 182549, 2020.

[10] S. Frayle Perez, «Integración de agentes deliberativos en la plataforma SPADE: Desarrollo de pGomas,» 2019.

[11] Django Software Foundation, «Django: The web framework for perfectionists with deadlines,» 2005-2023. [En línea]. Available: https://www.djangoproject.com/.

[12] Y. Díaz González y Y. Fernández Romero, «Patrón Modelo-Vista-Controlador,» *TELEMÁTICA,* vol. 11, nº 1, pp. 47-57, 2012.

[13] D. Bakken, «Middleware,» *Encyclopedia of Distributed Computing,* vol. 11, 2001.

[14] Hostinger, «¿Qué Es AJAX Y Cómo Funciona?,» 11 01 2023. [En línea]. Available: https://www.hostinger.es/tutoriales/que-es-ajax.

[15] C. Gómez Gil, «Objetivos de Desarrollo Sostenible (ODS): una revisión crítica.,» *Papeles de relaciones ecosociales y cambio global*, vol. 140, nº 1, pp. 107-108, 2018.

ANEXO I: Objetivos de desarrollo sostenible

Grado de relación del proyecto con los Objetivo de Desarrollo Sostenibles:

Objetivos de Desarrollo sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad ODS.**	**X**			
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**				**X**
ODS 10. **Reducción de las desigualdades.**				**X**
ODS 11. **Ciudades y comunidades sostenibles.**				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 metas globales adoptadas por los 193 Estados miembros de las Naciones Unidas, en septiembre de 2015. Los ODS tienen como objetivo abordar una amplia gama de desafíos mundiales, como la pobreza, el hambre, la salud, la educación, etc. Con el fin de promover un desarrollo sostenible a nivel social, económico y ambiental. [15]

El trabajo presentado está directamente relacionado con el ODS 4, Educación y calidad. Como se ha mencionado varias veces, este proyecto ofrece una herramienta a los estudiantes para facilitar las tareas que se realizan durante las sesiones prácticas en las asignaturas relacionadas con los agentes inteligentes.