Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

BOT PARA TELEGRAM QUE EJECUTE AVENTURAS CONVERSACIONALES

Alumno: Manuel Jesús García Quesada

Tutor: Arturo Montejo Ráez
Dpto: Departamento de Informática

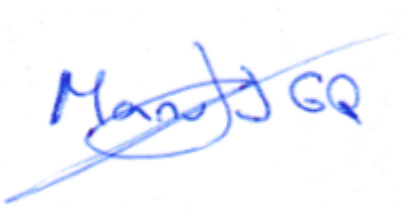
Septiembre, 2017

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Arturo Montejo Ráez, tutor del Proyecto Fin de Grado titulado: Bot para Telegram que ejecute Aventuras Conversacionales, que presenta Manuel Jesús García Quesada, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2017

El alumno: Los tutores:



Manuel Jesús Garcia Quesada Arturo Montejo Ráez

Índice

Índice de figuras

Índice de tablas

Índice de abreviaturas

API - Application Programming Interface / Interfaz de Programación de Aplicaciones

MTProto - Mobile Transport Protocol / Protocolo de Transporte Móvil

Nick - Nickname / Alias o seudónimo

HTTP - Hypertext Transfer Protocol / Protocolo de Transferencia de Hipertexto

URL - Uniform Resource Locator / Localizador de Recursos Uniforme

JSON - JavaScript Object Notation / Notación de Objetos de JavaScript

AES - Advanced Encryption Standard / Estándar de cifrado avanzado

XMPP - Extensible Messaging and Presence Protocol / Protocolo Extensible de Mensajería y Comunicación de Presencia

ID – Identifier / Identificador

SMS - Short Message Service / Servicio de Mensajes Cortos

SHA - Secure Hash Algorithm / Algoritmo de Hash Seguro

XML - Extensible Markup Language / Lenguaje de Marcado Extensible

UML - Unified Modeling Language / Lenguaje Unificado de Modelado

IDE - Integrated Development Environment / Entorno de Desarrollo Integrado

SQL - Structured Query Language / Lenguaje de Consulta Estructurada

IP - Internet Protocol / Protocolo de Internet

HTML - HyperText Markup Language / Lenguaje de Marcas de Hipertexto

1. Introducción

Una Aventura Conversacional es un género de videojuegos, más común en ordenadores, que nació en 1975, cuando Will Crowther escribió la Aventura Conversacional Adventure, originalmente conocida como ADVENT. Estos tipos de videojuegos tuvieron su época dorada en la década de los 80 y principio de los 90, aunque actualmente se siguen desarrollando videojuegos de este estilo por usuarios aficionados a las Aventuras Conversacionales, llegando incluso a ganar algunos de estos juegos premios a la mejor ficción interactiva en los últimos años [1].

Estos tipos de juegos se caracterizan por su sencillez, ya que se componen únicamente de texto con el cual se describe la situación o el lugar en la que él jugador se encuentra. A su vez, el jugador debe teclear la acción a realizar. El juego interpreta la entrada en lenguaje natural, lo cual le lleva a una nueva situación o lugar, y así sucesivamente, hasta completar el juego.

Estos juegos son creados bajo un lenguaje de programación especifico llamado Inform. Este lenguaje fue creado en 1993 por Graham Nelson, y se caracteriza por permitir desarrollar Aventuras Conversacionales para las máquinas virtuales Z-code o Glulxe. Otras de las características destacables de este lenguaje es que está completamente basado en los principios del lenguaje natural, algo que hace que este lenguaje sea el mejor a la hora de querer desarrollar una Aventura Conversacional [2].

Hoy en día este lenguaje sigue en desarrollo. Su última versión, Inform 7, incluye su propio entorno de desarrollo personalizado, además de permitir la creación de Aventuras Conversacionales completamente en español.

[1] Aventura conversacional - https://es.wikipedia.org/wiki/Aventura_conversacional
[2] Inform - https://en.wikipedia.org/wiki/Inform

1.1 Motivación del Proyecto

Hoy en día las comunidades de aficionados dedicadas a este tipo de videojuegos siguen activas por todo el mundo. A pesar de seguir en activo, es escaso el número de personas que siguen jugando este tipo de juegos, ya sea por el desconocimiento de la existencia de este tipo de videojuegos, en el caso de la gente joven, o porque ha quedado en el olvido tras la llegada de los videojuegos en 3D o las aventuras graficas.

Por ello, el fin de este Trabajo fin de Grado es rescatar los videojuegos de Aventuras Conversacionales clásicos y, junto a las Aventuras Conversacionales que hoy en día se siguen creando, crear una biblioteca para que cualquier persona que tenga a su alcance un dispositivo móvil o un ordenador pueda probar y jugar este tipo de juegos sin tener que realizar instalaciones y configuraciones de software que suponga una dificultad para el usuario.

De esta forma, nuestro objetivo es el de dar a conocer este tipo de videojuegos al mayor publico posible, de forma que las Aventuras Conversacionales puedan volver atrás en el tiempo y vivir de nuevo su época dorada. Además, las personas que siguen jugando a este tipo de videojuegos tendrán la oportunidad de poder jugar desde un dispositivo móvil, lo cual le permite disfrutar de sus videojuegos preferidos en cualquier lugar y en cualquier momento.

1.2 Objetivos

El objetivo de este Trabajo fin de Grado es el de diseñar e implementar un Bot para la plataforma de mensajería móvil Telegram el cual permita a los usuarios jugar a una Aventura Conversacional de forma fácil y cómoda. Este Bot hará de intermediario entre el usuario y la maquina Glulxe que ejecutará los distintos juegos desde un servidor.

La elección de hacer este Trabajo fin de Grado sobre un Bot de Telegram recae en la facilidad con la que un usuario puede interactuar con un Bot, ya que se interactúa de la misma forma que una persona habla diariamente con sus conocidos a través de plataformas para dispositivos móviles como WhatsApp o la ya nombrada Telegram, entre muchas otras.

Además de la facilidad con la que se interactúa, hay que añadir también la API que Telegram tiene para poder desarrollar Bots de cualquier tipo, ya que esta es de las más completas y documentadas que existe en cuanto a creación de Bots para este tipo de plataformas.

1.3 Definición del contexto del proyecto

1.3.1 Bots y API de Telegram

Telegram Messenger es un servicio de mensajería por Internet desarrollado desde el año 2013 por los hermanos Nikolai y Pavel Durov. El servicio está enfocado en la gestión de mensajes de texto y multimedia. Inicialmente fue empleado para teléfonos móviles y el año siguiente para multiplataforma [3].

Un Bot es un usuario autónomo el cual es controlado por botones interactivos, secuencias de texto o inteligencia artificial. A diferencia de los usuarios corrientes, los Bots se encargan de ofrecer diferentes servicios. Entre los servicios que se puede realizar con un Bot en Telegram destacan los comandos avanzados de sólo texto, la administración de canales y grupos, la posibilidad de compartir, crear encuestas en tiempo real, jugar a juegos e incluso pasarelas de pago [4]. Todos estos servicios usan el protocolo móvil MTProto.

[3] Telegram Messenger - https://es.wikipedia.org/wiki/Telegram_Messenger
[4] Telegram Bot API - https://es.wikipedia.org/wiki/Telegram_Bot_API

La API para desarrollar Bots de Telegram está disponible en la mayoría de los principales lenguajes de programación, lo que permite que cualquiera con los conocimientos básicos de programación pueda probar a crear su propio Bot. Además, cada Bot posee un token único que sirve para identificar a ese Bot dentro de los servidores de Telegram.

Además, esta API nos da acceso a prácticamente todo tipo de información sobre los usuarios o grupos. Esta información se estructura en distintos tipos de datos, entre los que destacan:

- **Usuario:** nos da toda la información acerca de un usuario de Telegram que esté usando nuestro Bot, como su Nick, su identificador único, su nombre y apellidos y el idioma que usa en Telegram.

- **Chat:** nos da todo tipo de información acerca de un chat en el que este el Bot. Este chat puede ser privado (un usuario hablando a solas con el Bot) o un grupo (varios usuarios interactuando con el Bot simultáneamente). Entre la información que nos da, destaca su identificador único, que tipo de chat es (grupo, supergrupo o canal), nombre del chat en el caso de que sea un grupo, etc.

- **Mensaje:** cada uno de los mensajes que se envían por un chat en el que se encuentra un Bot permite a este sacar bastante información acerca del mensaje, como su identificador, la fecha y hora a la que se mandó, quien lo mandó, chat al que pertenece, el contenido del mensaje, si el mensaje fue reenviado desde otro chat, etc.

- **ReplyKeyboardMarkup:** se trata de un teclado personalizado con distintas opciones que actúa como un menú para seleccionar una opción. Este teclado se compone de un array de KeyboardButtons.

- **KeyboardButtons:** este tipo de dato almacena la información relativa a cada uno de los botones que se crean cuando un Bot hace uso de teclados

personalizados para pedirle al usuario que escoja una opción. Estos botones almacenan básicamente el texto relativo a esa opción.

Existe muchos tipos de datos más aparte de los aquí comentados dentro de esta API, como Localizaciones, Stickers, Videos, Notas de Voz, Audios, entre muchísimos más. Me he limitado a explicar los tipos de datos a los que nuestro Bot hará uso.

Ahora vamos a explicar el funcionamiento de un Bot, el cual es sencillo de entender: un Bot puede tener Handlers o Webhooks.

Un handler actúa como un filtro, cada vez que el Bot reciben un mensaje, este pasa por los distintos handlers que el Bot tenga configurados y en el orden en el que estos estén dispuestos. Si uno de estos handlers o filtro da positivo (es decir, el mensaje cumple unas características específicas) el Bot comienza a procesar ese mensaje de alguna forma específica con el fin que él tenga ese handler (Por ejemplo, un handler que detecte cuando un usuario envía la palabra hola y haga que el Bot responda también con un saludo).

Un Webhook, al contrario que un handler, no funciona dentro de los chats de Telegram, sino que lo hace fuera. Estos Webhooks permiten al Bot recibir peticiones HTTP POST que contenga información estructurada en un JSON. Estas peticiones HTTP se enviarán a una URL especifica que tenga el Bot en cuestión configurada en un Webhook.

En nuestro caso, no hemos visto necesario el uso de Webhooks en nuestro Bot, ya que la iteración es siempre usuario-Bot a través de Telegram, y no desde fuera de esta plataforma.

1.3.2 MTProto

El protocolo de transporte móvil, abreviado como MTProto, es el nombre que recibe el protocolo de datos de la aplicación Telegram Messenger. Este protocolo está enfocado en la multisesión multiplataforma y el transporte de archivos sin

importar su formato o capacidad. El tráfico tiene dos tipos de cifrados, ambos con AES de base.

A diferencia del protocolo XMPP, MTProto genera una ID por cada dispositivo para el usuario, y que de esta forma el usuario tenga que hacer una autentificación en dos pasos. Estas ID se envían al dispositivo que el usuario tenga configurado como principal. Esta configuración del dispositivo principal se realiza mediante SMS o una llamada telefónica una única vez.

Es por esto por lo que el sistema de mensajería de Telegram es uno de los más seguros, además de que incluye un tercer tipo de chat, el chat secreto, los cuales aumentan más aun su seguridad, cifrando cada uno de los mensajes de extremo a extremo bajo el algoritmo SHA-1 y un cifrado XOR de 128 bits, además de una renovación de claves a medida que pasa el tiempo [5].

1.3.3 JSON

JSON es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente [6].

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON.

1.4 Estudio de soluciones existentes

La solución más vulgar es coger directamente la maquina Glulxe encargada de ejecutar los juegos, instalarla en un ordenador, y ponerte a jugar. Esto no se acerca en nada a lo que nosotros queríamos, ya que el objetivo como se ha dicho

[5] MTProto - https://es.wikipedia.org/wiki/MTProto
[6] JSON - https://es.wikipedia.org/wiki/JSON

antes es el de facilitar al usuario la necesidad de tener que instalar y configurar software, además de tener que buscar los distintos juegos que quieras jugar.

En internet se pueden encontrar algunas soluciones, como implementaciones web cuyo funcionamiento es parecido al de este Trabajo fin de Grado: una página web que actúa como interfaz para que el usuario juegue a una aventura conversacional que se está ejecutando en el mismo servidor que la web mediante la maquina Glulxe que es la encargada de hacer funcionar este tipo de juegos.

Existe también otra implementación del tipo cliente-servidor, el cual trabaja con pasos de mensajes JSON. Esta solución es la que más se aproxima a nuestro problema ya que, como se dijo anteriormente, el Bot actuará de intermediario entre el usuario y la maquina Glulxe que estará ejecutándose en un servidor, por lo que en la implementación cliente-servidor nuestro Bot sería el cliente.

Además de lo anteriormente citado, como se comentó anteriormente, la API de Bots de Telegram guarda los distintos tipos de datos (Chat, mensaje, etc.) con la estructura de un JSON, por lo que esa implementación cliente-servidor con pasos de mensajes JSON es lo que más se asemejaba a la solución de este Trabajo fin de Grado.

Para la resolución del problema de este Trabajo fin de Grado hemos partido de esta última implementación cliente-servidor, y basándonos en ella, hemos ido adaptándola a las limitaciones que tienen los Bots de Telegram

1.5 Perfil del Usuario

Este Bot va dirigido a todas aquellas personas que tengan a su alcance un dispositivo móvil o un ordenador en el que tengan instalado la aplicación de Telegram y les guste los videojuegos en general.

Hoy en día la mayoría de las personas tienen a su disposición un dispositivo móvil o un ordenador con las características suficientes como para tener instalado Telegram,

por lo que prácticamente quien quiera puede tener acceso a este Bot y de esa forma podrá probar y jugar una Aventura Conversacional.

De esta forma, los usuarios jóvenes a los que les gustan los videojuegos de las nuevas generaciones pueden conocer este tipo de juego, e incluso aficionarse a ellos. Por otra parte, habrá usuarios que conocían y jugaron a estos juegos en su época dorada, y gracias a este Bot puedan revivir esos juegos con los que pasaron mucho tiempo. Otro tipo de usuario puede ser aquel que no es aficionado a los videojuegos y, sin embargo, este tipo de juegos le acabe gustando tras probarlos.

Además de lo anteriormente dicho, he de destacar que este proyecto da la posibilidad a las personas ciegas de poder jugar, en algunos casos por primera vez en su vida, a un videojuego, ya que los avances tecnológicos en los dispositivos móviles permiten leer y escribir a través del altavoz/micrófono del dispositivo móvil [7].

Resumiendo, este Bot va dirigido a todo el público que tenga acceso a un dispositivo móvil u ordenador con Telegram, ya que para mucha gente este género es desconocido y es posible que le acabe gustando.

1.6 Análisis de Requisitos

Tras explicar cuál es el objetivo de este Trabajo de fin de Grado, las soluciones ya existentes de las que partimos y al público al que va dirigido, vamos a especificar los requisitos del software que vamos a desarrollar:

1.6.1.1 Requisitos Funcionales

El Bot debe estar siempre a la espera de nuevas órdenes por parte de los usuarios. Si por alguna razón un juego falla y devuelve un error, o simplemente no devuelve nada, el Bot debe saber actuar ante esa situación sin que afecte a su funcionamiento.

[7] Android 4.1 Jelly Bean y la accesibilidad - https://elandroidelibre.elespanol.com/2013/03/especial-android-y-las-aplicaciones-para-personas-ciegas.html

Además, el Bot debe proporcionar toda la ayuda necesaria para que el usuario sepa en todo momento todas las opciones y comandos que tiene a su disposición para hacer un uso correcto del Bot. El Bot también proporcionara ayuda acerca de cómo se comienza un nuevo juego, como se juega, etc.

El Bot mostrara todos los textos anteriormente citados en dos idiomas, español e inglés, dejando a elección del usuario en que idioma prefiere que aparezcan los textos. Los textos de cada uno de los juegos no pueden ser traducidos, por lo que se debe indicar, junto al nombre de cada juego, el idioma en el cual esta creado.

El Bot debe tener un conjunto de comandos que permita al usuario realizar cada una de las cosas que estamos comentando (jugar, pedir ayuda, aprender a jugar, escoger idioma, etc.).

El Bot puede ser jugado en dos modos: un jugador (Chat privado) o multijugador (Chat de grupo)

1.6.1.2 Requisitos no Funcionales

El Bot debe tener una base de datos del tipo que sea en la que se guardará todos los datos relacionados con los usuarios y grupos de Telegram que hagan uso del Bot (Nombre, Nick, identificador único, estado actual de la partida, el idioma que haya elegido y si se encuentra eligiendo algo en un menú)

Además, el Bot tiene que tener a su disposición un Servidor que ejecute los diferentes videojuegos de Aventura Conversacionales. Este servidor forma parte de otro Trabajo fin de Grado.

1.7 Análisis del Sistema

Una vez que hemos visto todos los requisitos que necesitamos en nuestro Bot, podemos representar los diferentes casos de uso que podrá tener un usuario con este Bot:

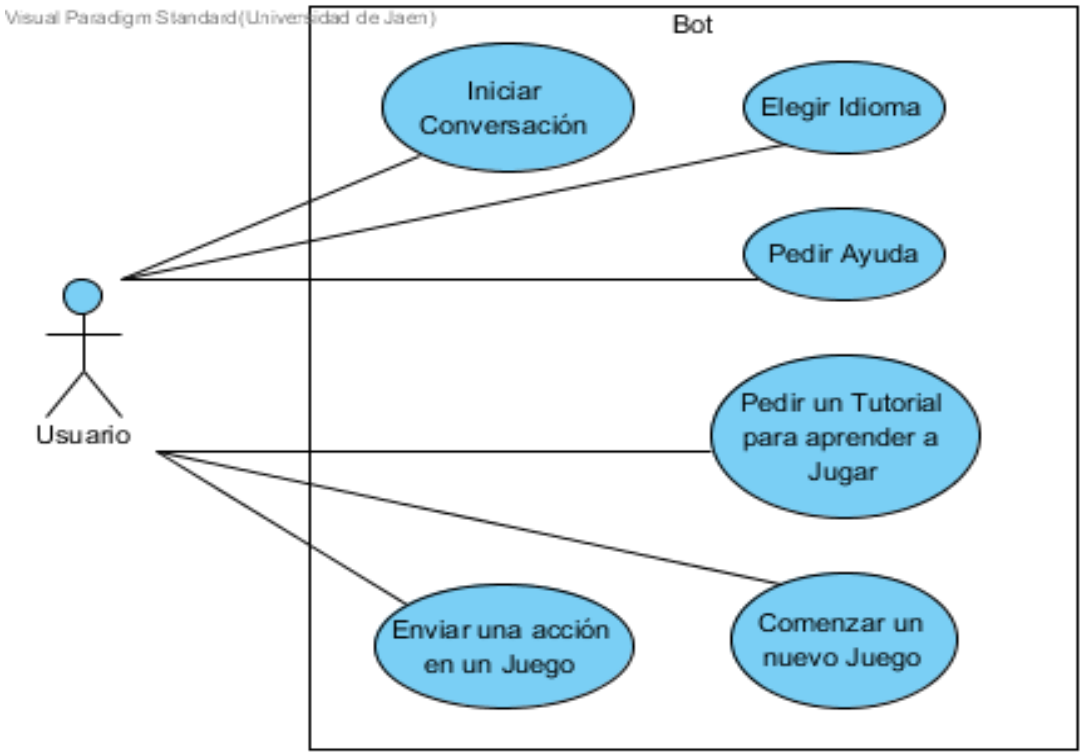


Figura 1.1 Casos de Uso del Usuario con el Bot

Como se puede observar en el diagrama UML de casos de uso anterior, el usuario dispondrá de una serie de opciones para interactuar con el Bot. Vamos a comentar cada una de estas más detenidamente.

- **Iniciar Conversación con el Bot:** es la primera acción que el usuario debe realizar para posteriormente interactuar con él realizando el resto de acciones. Una conversación con un Bot se inicia una única vez, a no ser que borres la conversación o detengas el Bot. Detener el Bot es una opción de Telegram la cual consiste en notificar al Bot que vas a dejar de usarlo de forma permanente (hasta que el usuario vuelva a iniciar la conversación). De

esta forma, el Bot dejara de estar pendiente por si ese usuario le enviaba algún comando.

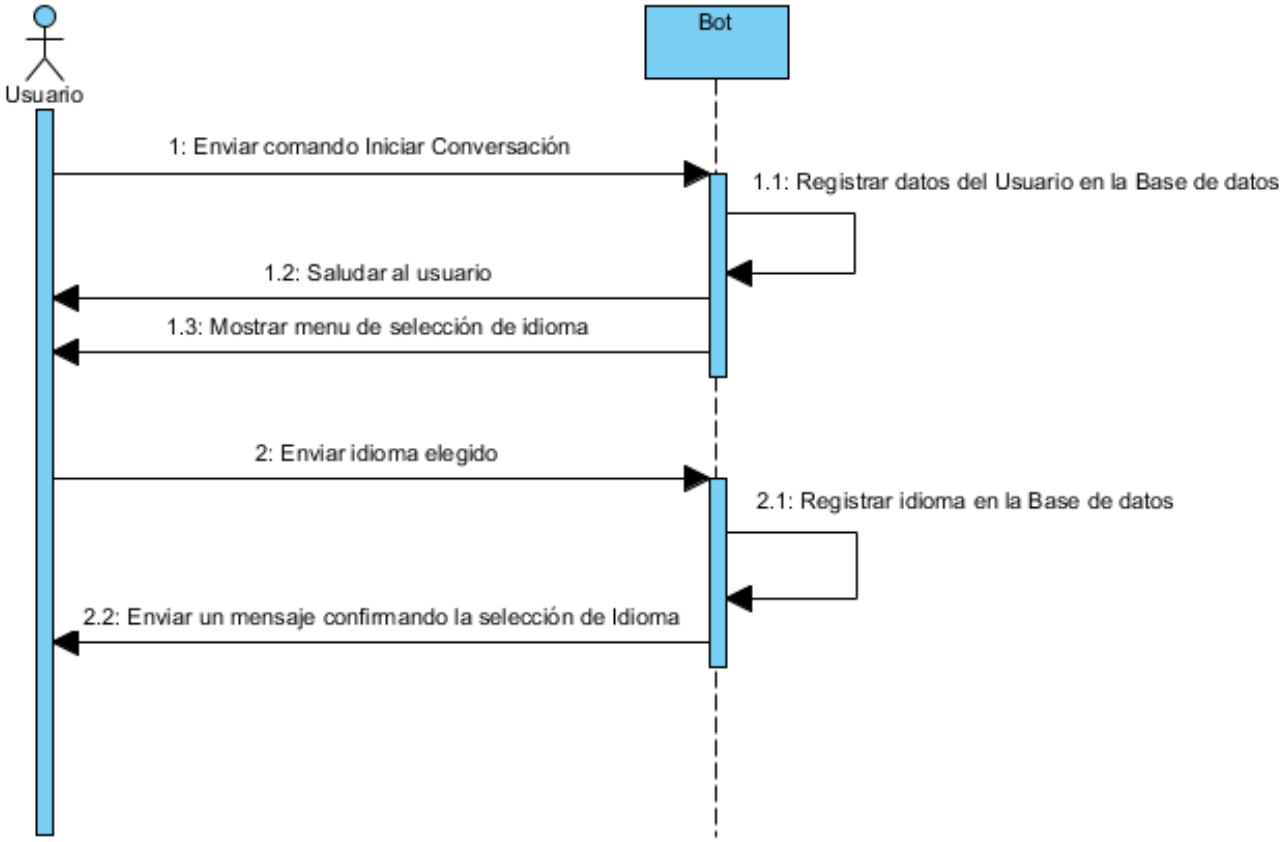


Figura 1.2 Diagrama de secuencia de la acción Iniciar Conversación

- **Elegir Idioma:** el usuario puede pedirle en cualquier momento al Bot que desea cambiar el idioma en el que se muestran los textos de este (excepto en los juegos, como se dijo anteriormente). Para ello, basta con que el usuario le envíe un comando que el Bot tenga configurado como comando para cambiar de idioma.

- **Pedir Ayuda:** si el usuario no sabe o se le ha olvidado como se usaba el Bot, podrá enviarle a este un comando pidiendo que le muestre todas las acciones que puede hacer el usuario.

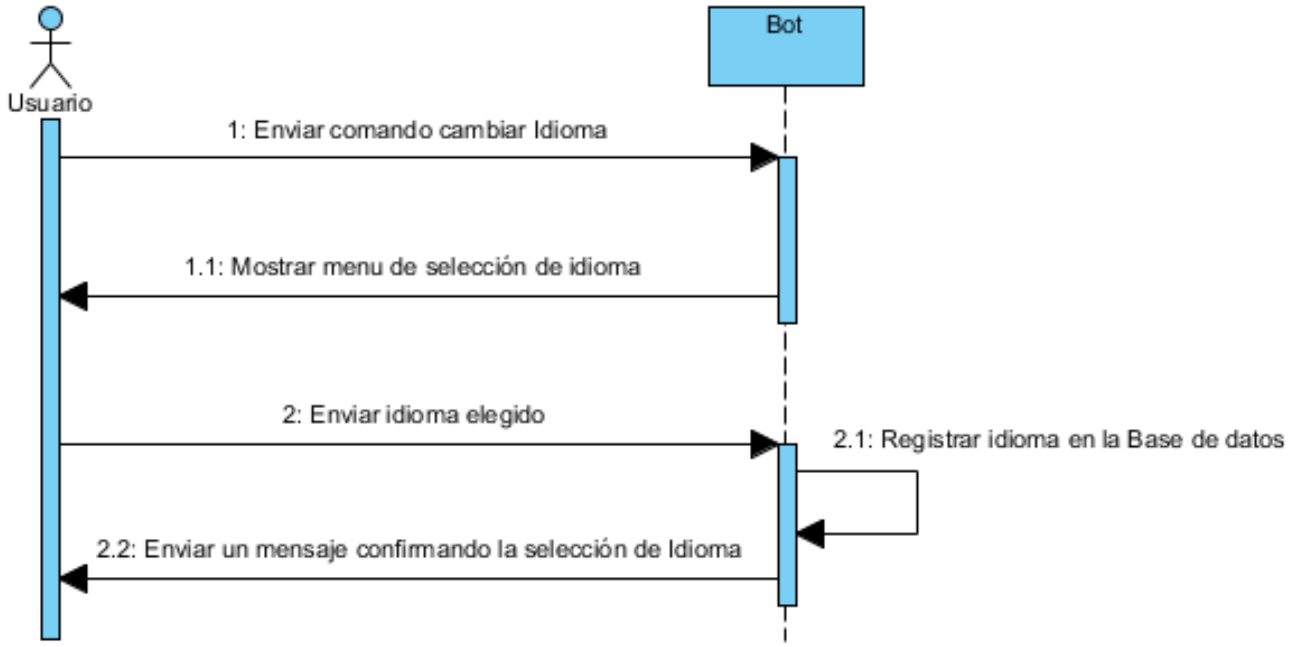


Figura 1.3 Diagrama de secuencia de la acción elegir Idioma

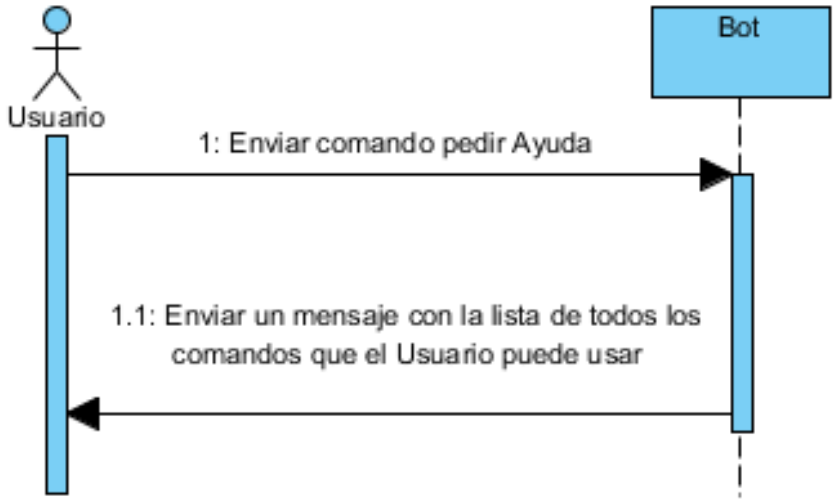


Figura 1.4 Diagrama de secuencia de la acción pedir Ayuda

- **Pedir un tutorial para aprender a Jugar:** al igual que la anterior acción, es posible que el usuario no sepa cómo se juega a las Aventuras Conversacionales. Por este motivo, el usuario tendrá a su disposición un comando con el cual pedirá al Bot que le envíe un tutorial completo sobre como jugar a cada una de las Aventuras Conversacionales.

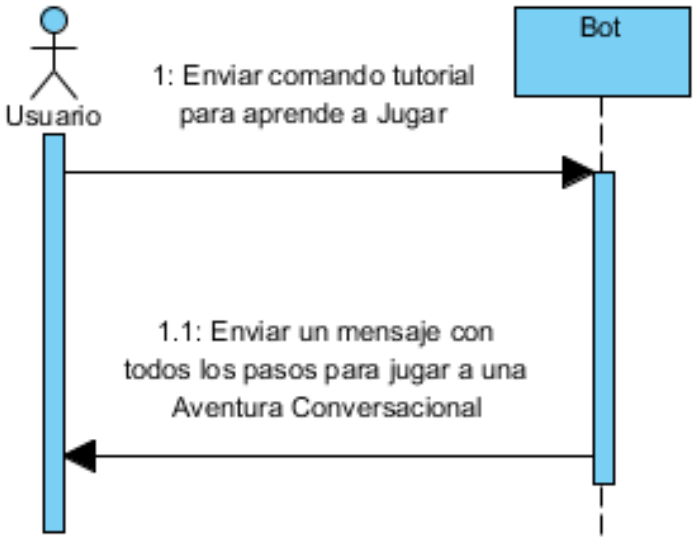


Figura 1.5 Diagrama de secuencia de la acción pedir un tutorial para aprender a Jugar

- **Comenzar un nuevo juego:** el usuario puede pedir al Bot que quiere jugar a un nuevo juego, y este le mostrará, siempre que el usuario no esté en ese momento jugando a una Aventura Conversacional, una lista con todos las Aventuras Conversacionales que tenga a su disposición para jugar.

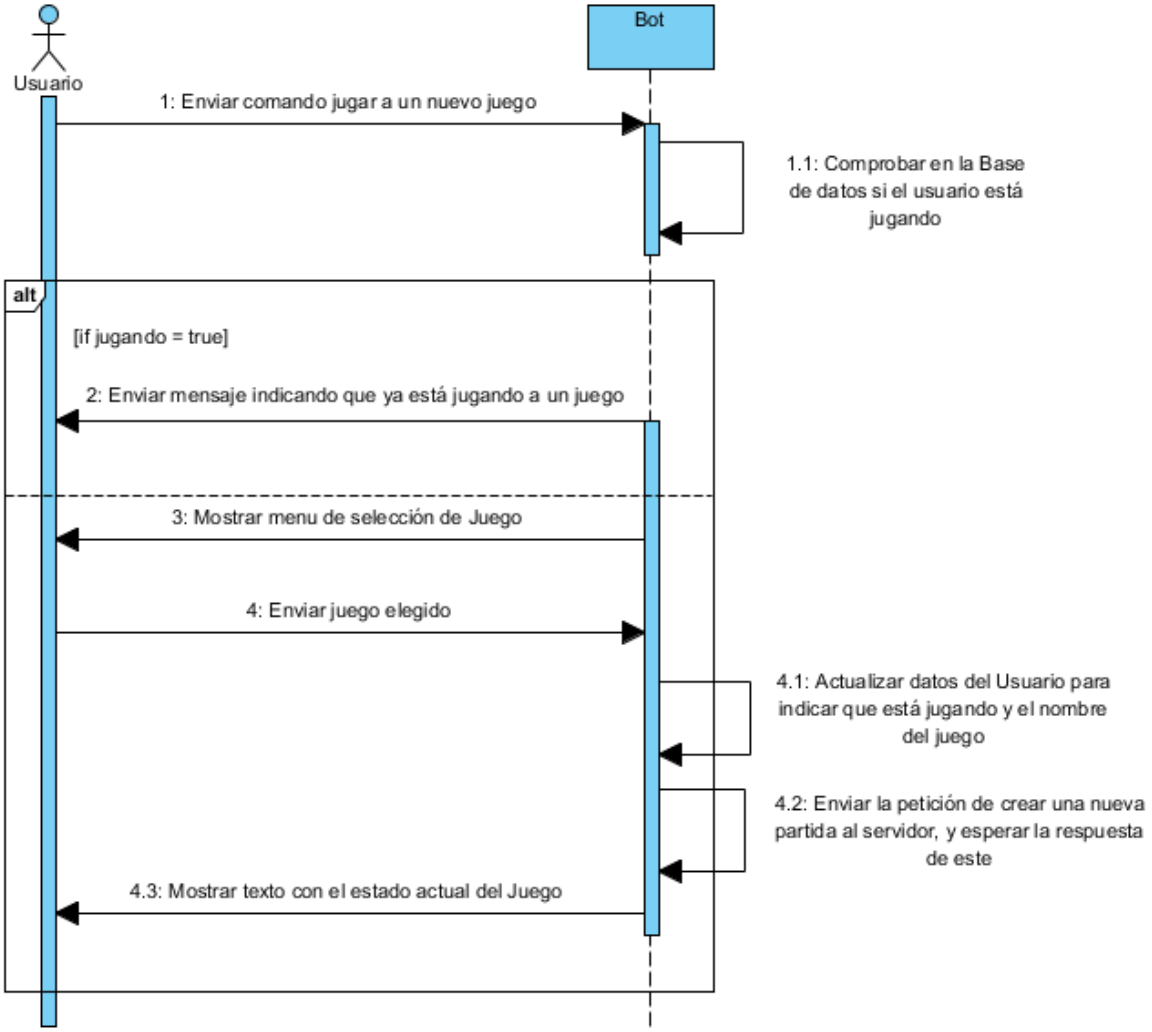


Figura 1.6 Diagrama de secuencia de la acción comenzar un nuevo juego

- **Enviar una acción en un juego:** una vez el usuario se encuentra jugando con una Aventura Conversacional, este debe escribir en el chat la acción que él piensa que es la correcta para avanzar en el juego. El Bot recogerá el texto que le envíe el usuario y se lo entregará al servidor donde se ejecutan las distintas Aventuras Conversacionales en una máquina virtual Glulxe.

  A pesar de que cada juego es totalmente distinto a los demás y, por lo tanto, el conjunto de verbos con acciones que admite para que el usuario avance

por el juego pueda cambiar, existe un conjunto de palabras claves que detectan todos los juegos. Estas palabras claves son:

- **Quit:** esta palabra se usa para salir y cerrar todos los juegos. Como en todos los videojuegos, si cierras la partida sin previamente haber guardado la misma, perderás todo el progreso de la misma. A diferencia de los videojuegos actuales, nuestro sistema no contempla un guardado automático de las partidas, por lo que recae en el usuario la labor de guardar partida antes de cerrar el juego, con la siguiente palabra. En algunos juegos en castellano la palabra es cambiada por **Salir**.

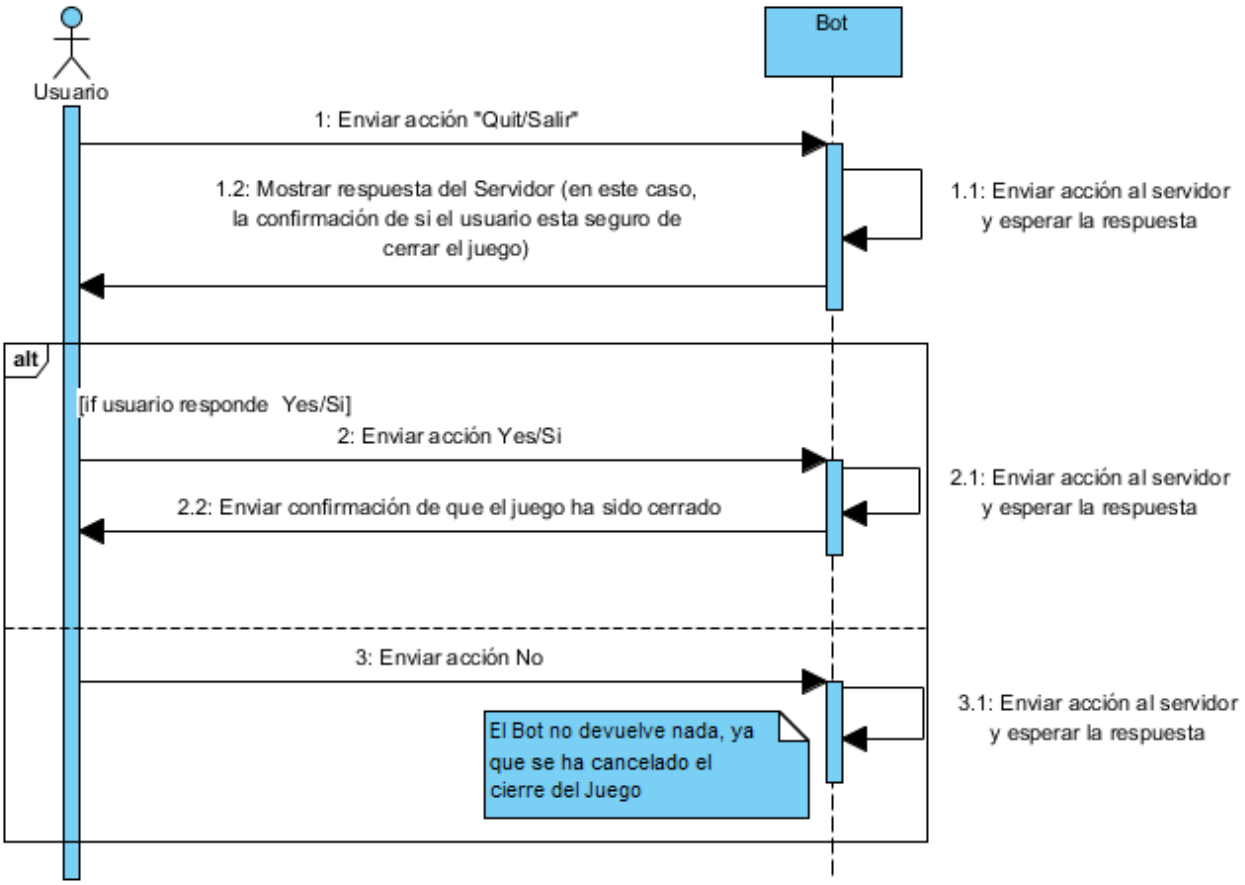


Figura 1.7 Diagrama de secuencia de la acción enviar acción especial Quit

- **Save:** esta palabra es la encargada de indicarle a Glulxe que queremos guardar el progreso de nuestra partida. Una vez se lo indiquemos, se guardará en un archivo cuyo nombre incluirá nuestra ID única de Telegram junto al nombre del juego. De esta forma, ningún otro usuario podrá sobrescribir nuestra partida. El Bot nos devuelve un mensaje indicándonos si la partida se ha guardado correctamente o no. En algunos juegos en castellano la palabra es cambiada por **Guardar**.

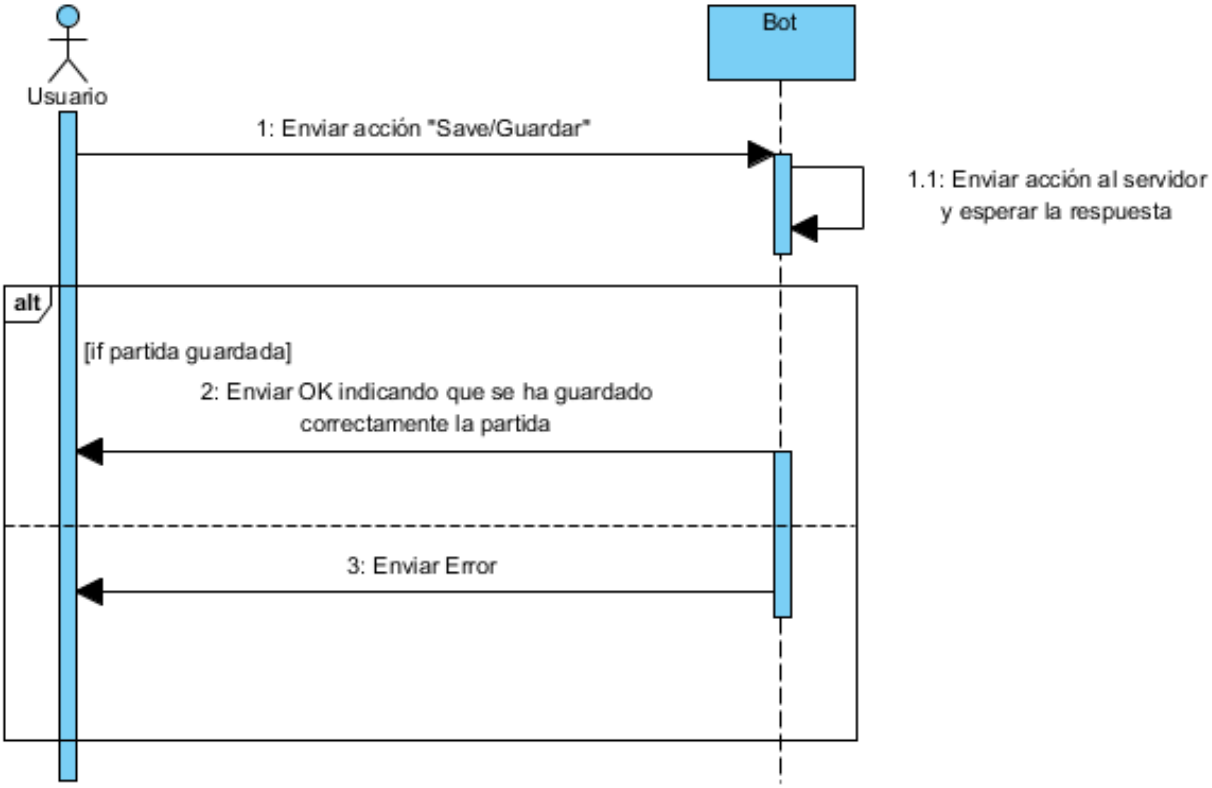


Figura 1.8 Diagrama de secuencia de la acción enviar acción especial Save

- **Restore:** con esta palabra indicaremos que queremos recuperar nuestra partida, siempre y cuando esta exista, y previamente le hayamos pedido al Bot iniciar el juego en cuestión. El Bot nos devolverá un mensaje indicándonos si se ha cargado o no la partida, pero no puede decirnos donde estábamos dentro del juego. En algunos juegos en castellano la palabra es cambiada por **Cargar**.

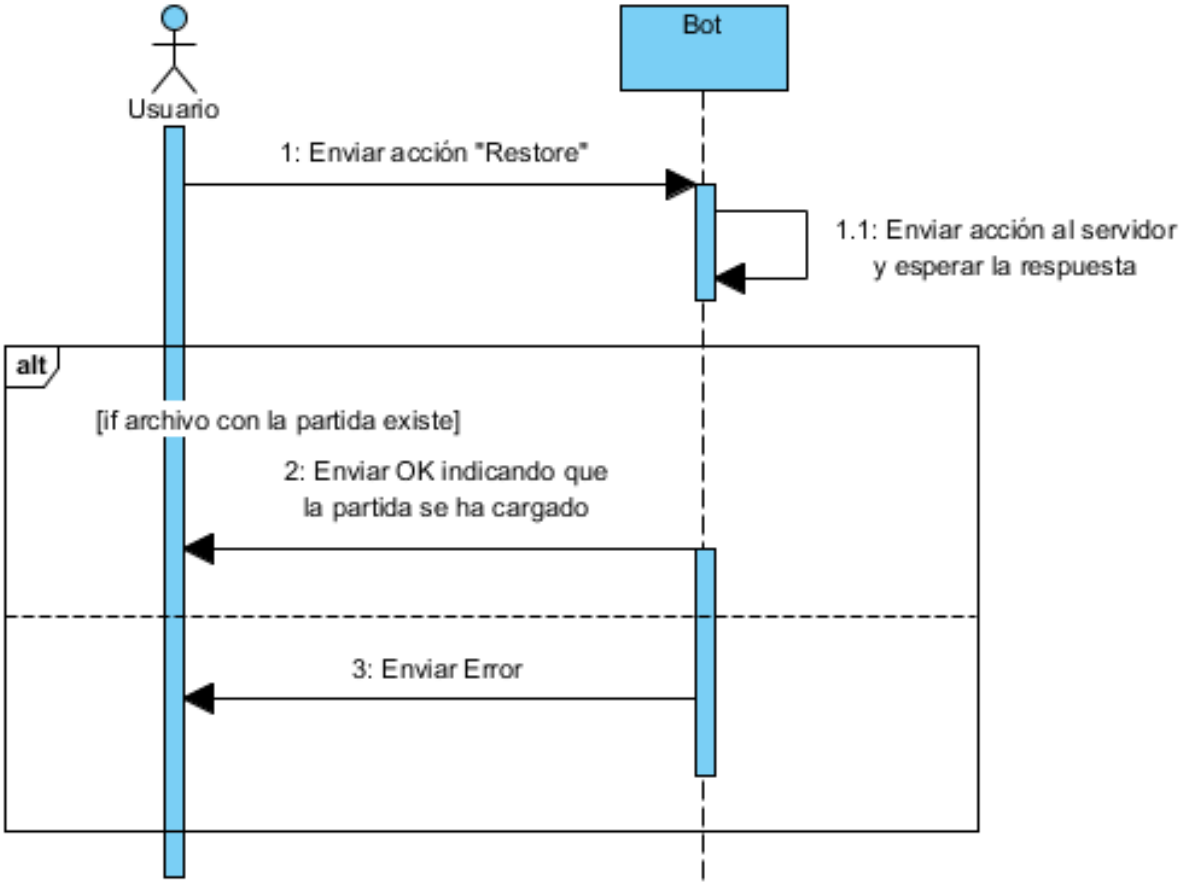


Figura 1.9 Diagrama de secuencia de la acción enviar acción especial Restore

- **Look:** esta palabra sirve para que el juego nos repita en qué lugar o situación nos encontramos. De esta forma, esta palabra se usa una vez se haya cargado la partida porque, como se comentó anteriormente, cuando cargamos la partida el Bot solo nos puede devolver un mensaje indicándonos si la operación se realizó correctamente. En algunos juegos en castellano la palabra es cambiada por **Mirar**.

Para el resto de acciones, el funcionamiento que sigue es el mismo que para el diagrama que más abajo aparece en la acción Look.

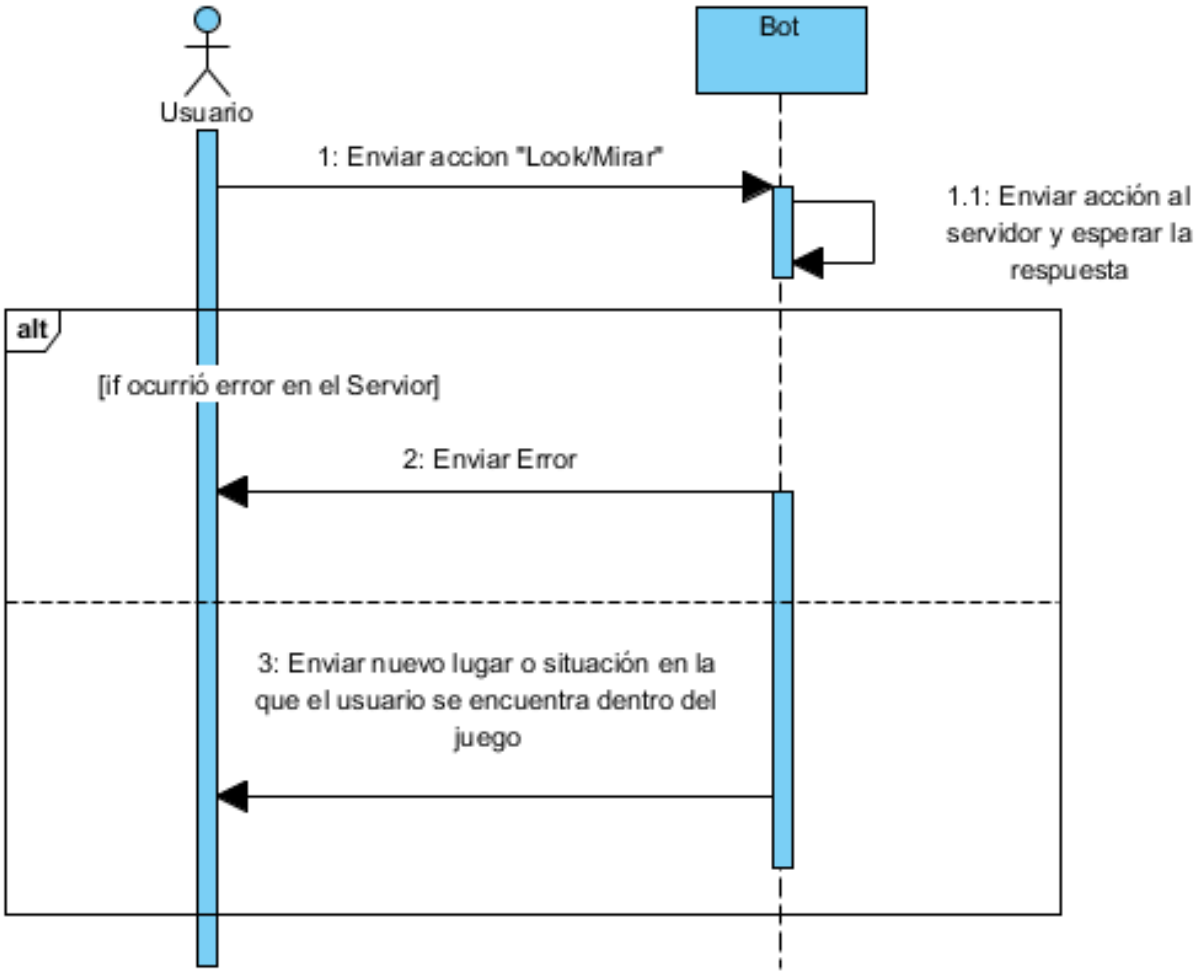


Figura 1.10 Diagrama de secuencia de la acción enviar acción especial Look / enviar acciones genéricas

2. Propuesta de solución

Una vez terminada la fase anterior de análisis del sistema a desarrollar, vamos a detallar cuales han sido las diferentes decisiones que se han ido adoptando a lo largo del desarrollo de la solución de este Trabajo fin de Grado.

2.1 Servidor

Como se comentó anteriormente, el Bot trabaja en conjunto con un Servidor, el cual forma parte de otro Trabajo de fin de Grado. La función de este servidor es ejecutar la máquina virtual Glulxe. Esta máquina virtual es la encargada de ejecutar las distintas Aventuras Conversacionales que el servidor tenga a su disposición. El Bot y el Servidor se comunican a través de mensajes HTTP POST y GET junto con el paso de información JSON, como los textos del juego o la acción que envía un jugador.

2.2 Planificación de las distintas fases del proyecto

Antes de pasar a comentar las distintas fases que se deben desarrollar en este Bot comentar que, al igual que se dijo anteriormente, este Trabajo fin de Grado abarca la mitad de un proyecto conjunto. La otra parte, que abarca otro Trabajo fin de Grado y que ha realizado otro alumno de la carrera, se centra en la implementación del servidor donde se ejecutan los juegos.

Por esto, la planificación de las distintas fases depende también de la parte del servidor, ya que, sin él, el Bot seria inutilizable. Habiendo hecho este inciso, pasamos a detallar como se ha ido planificando las diferentes fases que componen el desarrollo del Bot.

El proyecto se divide en tres fases: investigación, diseño y desarrollo.

2.2.1 Investigación

Una vez teníamos claro lo que teníamos que hacer, hay que dedicar una primera parte de la planificación del proyecto a investigar acerca del mundo de las Aventuras Conversacionales para entender su funcionamiento dentro de la máquina virtual Glulxe, ya que posteriormente hay que preparar el Bot para que sea capaz de enviarle a esta máquina virtual mensajes compuestos por JSON, al igual que tiene que ser capaz de interpretar los mensajes JSON que recibe por parte de Glulxe para mostrárselos correctamente al usuario.

Además de investigar acerca de las Aventuras Conversacionales, también hay que estudiar a fondo cómo funcionan los Bots y la API de Telegram. Una vez descrito rápidamente las distintas tareas, vamos a exponerlas de forma más clara con sus distintas iteraciones ordenadas cronológicamente:

- **Iteración 1:** duración de dos semanas

Prioridad	Tarea	Duración
8	Investigar acerca de la historia de las Aventuras Conversacionales	1 día
1	Estudiar el funcionamiento de las Aventuras Conversacionales	1 día
5	Investigar acerca de las comunidades relacionadas a las Aventuras Conversacionales que sigan en activo	1 día
6	Investigar acerca de la cantidad de juegos disponibles para la máquina virtual Glulxe	1 día

7	Investigar acerca de la historia de la máquina virtual Glulxe	1 día
2	Investigar acerca del funcionamiento de la maquina Glulxe	1 día
3	Estudiar como estructura la información interna que posteriormente muestra la máquina virtual Glulxe	1 día
4	Investigar acerca del formato JSON	1 día
9	Investigar acerca del lenguaje de programación Inform 7	1 día
10	Investigar acerca del funcionamiento de los Bots de Telegram	1 día
11	Estudiar toda la funcionalidad que nos da la API de Telegram	1 día
12	Estudiar los diferentes tipos de datos que se pueden manejar con la API de Telegram	1 día
13	Estudio de los distintos lenguajes de programación que soporta la API de Telegram, comparándolos	1 día
14	Revisión de que todas las tareas anteriores se han realizado correctamente y en el plazo indicado	1 día

Tabla 2.1 Listado de tareas de la iteración 1 de la fase de Investigación

- **Iteración 2:** duración de dos semanas

Prioridad	Tarea	Duración
1	Escoger el lenguaje de programación que mejor se adapte a nuestras necesidades	1 día
2	Estudio de los distintos entornos de desarrollo disponibles para el lenguaje escogido	1 día
3	Testeo de cada uno de los entornos encontrados, escogiendo el que mejor se adapte a nuestras necesidades	1 día
4	Escoger un sistema operativo para trabajar en el desarrollo del Bot	1 día
5	Crear un pequeño Bot en el que se testeen todas las funcionalidades de los Bots, y estudiar su comportamiento	1 día
6	Investigar acerca de las distintas APIs disponibles en el lenguaje escogido para el envío de mensajes HTTP cuyo contenido sean JSON	1 día
7	Escoger la API HTTP que mejor se adapte a nuestras necesidades, y realizar una prueba de conexión	1 día

8	Realizar una investigación acerca de las distintas plataformas y lenguajes para la creación de una Base de Datos y realizar una lista	1 día
9	Realizar una comparativa con todas las opciones de la lista, estudiando las ventajas e inconvenientes de cada uno	1 día
10	Escoger la mejor plataforma para la creación de una Base de Datos, atendiendo a la anterior comparativa, y realizar una simulación con una Base de Datos de prueba	1 día
11	Investigar acerca de los diferentes ejemplos que existen en internet de implementaciones cliente-servidor o aproximaciones a esta que usen Glulxe	1 día
12	Testear cada uno de los ejemplos que se encuentren y estudiar cómo funcionan	1 día
13	Escoger el ejemplo que mejor se adapte a nuestras necesidades, y estudiar la estructuración del cliente	1 día
14	Revisión de que todas las tareas anteriores se han realizado correctamente y en el plazo indicado	1 día

Tabla 2.2 Listado de tareas de la iteración 2 de la fase de Investigación

2.2.2 Diseño

Una vez hayamos investigado acerca de cómo funcionan Glulxe, los Bots y la API de Telegram, y además tengamos decidido que plataformas de desarrollo vamos a utilizar para el Bot y la base de datos, pasamos a la fase de diseño. Esta fase es la que más depende del servidor, ya que deberemos adaptarnos a su forma de funcionar. En esta fase, las tareas tienen una prioridad secuencial, ya que no debemos pasar a la siguiente tarea hasta no tener finalizadas las anteriores.

- **Iteración 1:** duración de dos semanas

Prioridad	Tarea	Duración
1	Diseño de las distintas tareas que deberá realizar el Bot y los distintos comandos que este tendrá	1 día
2	Diseño del logotipo que aparecerá en el chat de Telegram con el Bot	1 día
3	Diseño de los distintos handlers que tendrá el Bot en función de los comandos y la funcionalidad de este	1 día
4	Diseño de los distintos módulos en los que se estructurará el Bot	1 día
5	Diseño de la estructura interna de los módulos más simples del Bot (pedir ayuda, mostrar un tutorial para aprender a jugar y otra información adicional)	1 día

6	Diseño de la estructura interna del módulo de escoger idioma	1 día
7	Diseño de la estructura interna del módulo de iniciar conversación con el Bot	1 día
8	Diseño de la estructura interna del módulo para comenzar un nuevo juego	1 día
9	Diseño de la estructura interna del módulo para jugar y enviar acciones al juego	1 día
10	Diseño de la información necesaria a almacenar por parte de un usuario de Telegram	1 día
11	Diseño de la información necesaria a almacenar por parte de un grupo de Telegram	1 día
12	Diseño de la estructura de la Base de Datos	1 día
13	Comprobación de todo el diseño	1 día
14	Revisión de que todas las tareas anteriores se han realizado correctamente y en el plazo indicado	1 día

Tabla 2.3 Listado de tareas de la iteración 1 de la fase de Diseño

2.2.3 Desarrollo

Tras haber terminado el diseño de todo lo relacionado al Bot, es hora de ponernos con la última fase, el desarrollo. En esta fase debemos implementar toda la estructura del Bot y de la Base de Datos y, en conjunto con el Servidor, testear todo el funcionamiento. En esta fase, al igual que en la anterior de diseño, las tareas tienen una prioridad secuencial, ya que no debemos pasar a la siguiente tarea hasta no tener finalizadas las anteriores.

- **Iteración 1:** duración de dos semanas

Prioridad	Tarea	Duración
1	Implementación de la estructuración de la Base de Datos	1 día
2	Implementación de la arquitectura de conexión entre la Base de Datos	1 día
3	Testeo de la Base de Datos, comprobando que inserte y extraiga datos correctamente	1 día
4	Implementación base del Bot, registrándolo en Telegram y configurando todos sus comandos/handlers	1 día
5	Testeo inicial del Bot, comprobando que recibe mensajes desde diferentes dispositivos	1 día

6	Implementación de los módulos más simples del Bot, acorde a su diseño (pedir ayuda, mostrar un tutorial para aprender a jugar y otra información adicional)	1 día
7	Testeo de cada uno de los módulos implementados, intentando buscar posibles fallos	1 día
8	Implementación del módulo de selección de idioma y de inicio de la conversación con el Bot	1 día
9	Implementación y testeo de la arquitectura de conexión con el Servidor, mandando y recibiendo mensajes JSON	1 día
10	Implementación de un convertidor de datos JSON a texto normal para mostrarlo en el chat	1 día
11	Implementación del módulo de comenzar un nuevo juego y jugar, atendiendo al diseño previamente realizado	1 día
12	Traducción de todos los módulos del Bot tanto al inglés como al español	1 día
13	Testeo final del Bot, comprobando que todas sus características funcionan correctamente	1 día

14	Revisión de que todas las tareas anteriores se han realizado correctamente y en el plazo indicado	1 día

Tabla 2.4 Listado de tareas de la iteración 1 de la fase de Desarrollo

2.3 Estimación de costes

Atendiendo al anterior apartado de la planificación del proyecto, vamos a estimar los costes que supone el desarrollo de este, suponiendo que fuera a desarrollarse profesionalmente.

En primer lugar, el total de horas de trabajo que suponen el completo desarrollo del Bot. Teniendo en cuenta que la planificación anterior se compone de 56 días de trabajo, a 5 horas de trabajo por día, tendríamos una duración de 280 horas de trabajo. Poniendo un coste de 9€ la hora de trabajo por persona, y considerando este proyecto de dificultad media y que dos personas no pueden realizar una misma tarea, tendríamos un precio final de 3780€ a repartir en función del número de horas que trabaje cada una de las personas que trabajen en el desarrollo del Bot.

Coste: 280 horas x 9€/hora = 2520€
Complejidad del proyecto: 50%(Dificultad Media) de 2520€ = 1260€
Coste Total = 2520€ + 1260€ = **3780€**

Figura 2.1 Cálculos de la estimación de costes

Además del coste de las personas, hay que añadir el coste de las herramientas con las que se trabajará en el desarrollo, además del coste del hosteo del Bot para que este esté disponible las 24 horas del día.

En cuanto a las herramientas de trabajo, cada persona que participe en el desarrollo deberá tener a su disposición un ordenador con los requisitos suficientes para permitir a la persona desarrollar sin problemas. Sabiendo esto, el coste será de 300€ por ordenador/persona.

En cuanto al hosteo, hoy en día existen muchas compañías que ofrecen servicios de hosteo. El coste medio de un hosteo es de 5€ al mes, por lo que es bastante más económico que hacer nosotros mismos el hosteo (gastos de luz, mantenimiento, etc.).

2.4 Metodología utilizada

La metodología de trabajo adoptada durante el desarrollo de este Trabajo fin de Grado ha sido el de programación extrema.

La programación extrema es una metodología de desarrollo ágil que tiene como principal objetivo aumentar la productividad a la hora de desarrollar un proyecto software. Da prioridad a los trabajos que dan un resultado directo y en los cuales se reduce la burocracia que pueda existir en el entorno de trabajo [8].

De esta forma, a pesar de tener una planificación inicial de cómo debería transcurrir el desarrollo del proyecto, podíamos adaptarnos a las distintas necesidades que fueran surgiendo a raíz de las distintas pruebas que fuéramos realizando sobre el Bot.

2.5 Análisis de las herramientas

Una vez teníamos claro todo el trabajo a realizar y que metodología de trabajo seguir para la creación del Bot es hora de pensar con qué hacerlo, y en qué ámbito. Por ello, en este apartado vamos a comentar que entorno, lenguaje de programación, sistema operativo, base de datos, APIs y librerías hemos utilizado para el desarrollo del proyecto:

- **Sistema operativo:** el sistema operativo que decidí usar para trabajar en este Trabajo fin de Grado es el que siempre he usado durante toda la carrera, Windows. En concreto he utilizado su última versión, Windows 10.

[8] Mario Pérez Esteso (2017) - Programación extrema: qué es y principios básicos - https://geekytheory.com/programacion-extrema-que-es-y-principios-basicos

La elección de este sistema operativo recae en que es el más completo en cuanto a herramientas para desarrolladores en prácticamente cualquier lenguaje de programación, además de que, como dije antes, es el sistema operativo que he usado siempre y por ello, es el que mejor domino. De esta forma, no tengo que perder tiempo en el aprendizaje del manejo de un sistema operativo nuevo.

- **Lenguaje de programación y entorno:** una de las grandes ventajas que tenía a la hora de desarrollar un Bot para Telegram es, como se comentó anteriormente, la API para desarrolladores que tiene, ya que esta está disponible en la gran mayoría de lenguajes de programación. Tras echar un vistazo a todos los lenguajes de programación que tenía para elegir, me decante por utilizar Python.

  La razón por la cual decidí utilizar Python es por ser el lenguaje más utilizado por todos los desarrolladores del mundo a la hora de programar un Bot de Telegram, lo que quiere decir que podría encontrar ejemplos de funcionamiento de Bots fácilmente, además de poder encontrar rápidamente respuesta a las dudas que podrían surgirme a lo largo del desarrollo gracias a la gran comunidad de desarrolladores que hay implementando Bots de Telegram con Python.

  Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma [9].

  Además de todo lo anterior, hay que añadir que Python es un lenguaje que ya conocía y con el que trabajé antes de este proyecto por lo que, al igual que

[9] Python - https://es.wikipedia.org/wiki/Python

con el sistema operativo, no era necesario gastar tiempo en el aprendizaje de un nuevo lenguaje de programación.

En cuanto al entorno de desarrollo, fue una decisión que me tomó más tiempo, ya que probé varios entornos de desarrollo de Python, pero no me terminaban de convencer ninguno. Comencé probando un entorno de desarrollo más complejo y completo llamado Komodo.

Según expertos, Komodo es el mejor entorno de desarrollo para trabajar con Python [10] [11], pero el problema que encontré fue que era tan complejo de configurar que decidí descartarlo, ya que era una pérdida de tiempo innecesaria. Después probé con Visual Studio, ya que lo estaba utilizando para desarrollar otros proyectos en el lenguaje C++ y entonces no requería tiempo en tener que instalarlo.

Sin embargo, tras configurarlo todo para que detectara Python, daba problemas con la indentación del código, por lo que al final me acabe decantando por el entorno de desarrollo oficial de Python. El único inconveniente que tenía este era su simplicidad, ya que se trata de un editor de texto preparado para el formato del lenguaje Python. Aun así, no necesitaba configuración y fue el único que no me dio problemas a la hora de probarlo, por lo que me quedé finalmente con él.

- **Base de Datos:** tras realizar un estudio acerca de las distintas opciones que tenía a mi disposición, me decante por FireBase.

  FireBase es una plataforma de desarrollo creada por Google recientemente. Esta plataforma tiene a disposición de los desarrolladores distintos servicios, entre los que destacan Hosting, Almacenamiento, Base de Datos, etc. [12]

[10] Komodo IDE: The best IDE for web and mobile app development - https://www.activestate.com/komodo-ide
[11] Los 10 mejores IDE´s para Python (2016) - http://notasinformaticas.com/los-10-mejores-ides-para-python/
[12] Firebase - https://en.wikipedia.org/wiki/Firebase

Centrándonos en la Base de Datos, las ventajas que tiene FireBase, respecto a al resto de plataformas para la creación de Bases de Datos, es que tiene un plan gratuito que te permite tener tu Base de Datos hosteada por ellos de forma gratuita, con unos límites de conectividad y almacenamiento. Sin embargo, estos límites en el plan gratuito de FireBase eran suficientes para el tipo de información que teníamos pensado almacenar.

Otra gran ventaja que tiene la Base de Datos de FireBase es que la información no se estructura en Tablas como lo hacen las Bases de Datos clásicas que usan SQL, sino que la información es estructurada en un único JSON, lo que hace que la manipulación de la información sea muy simple.

Como íbamos a trabajar con JSON en el paso de mensajes Bot-Servidor, que la Base de Datos también se estructurara como un JSON simplificaba mucho el trabajo, ya que en todo momento íbamos a trabajar únicamente con JSONs.

- **APIs y librerías utilizadas:** además de la API de Telegram ya explicada anteriormente, hemos tenido que buscar diferentes APIs que satisficieran las necesidades del Bot:

    - **Bot:** tras estudiar las distintas adaptaciones a la API oficial de Telegram en Python, me decanté por unas de las más conocidas y usadas: pyTelegramBotAPI [13].

      Esta API se caracteriza por su sencillez, ya que simplifica la API oficial de Telegram, abstrayendo todo el tema de consultas por URL y conexión con Telegram. Es por esto por lo que esta API es una de las más populares que se pueden encontrar por internet, ya que te permite programar tu Bot de Telegram fácilmente.

[13] eternnoir (2015) pyTelegramBotAPI - https://github.com/eternnoir/pyTelegramBotAPI

- **Conexión con el Servidor:** para realizar el paso de mensajes con el Bot necesitaba una API que realizara peticiones HTML POST y GET, además de poder incluir datos estructurados en un JSON junto a la petición.

  Tras buscar diferentes APIs que cumplieran estos requisitos y que fuera compatible con Python, me decante por Requests. Esta librería para Python se caracteriza por quitar todas las complicaciones que conllevan trabajar con HTTP en Python, haciendo que la integración con servicios web sea transparente [14].

  De esta forma, con una simple llamada a una función get o post, podemos enviarle a cualquier servidor una petición HTTP que incluya datos JSON, justo lo que necesitábamos para nuestro problema.

- **Traducciones:** para internacionalizar los textos de nuestro Bot teníamos que encontrar una API o librería en Python para cargar archivos con las traducciones fácilmente e internacionalizar nuestro software. Rápidamente encontramos pygettext, el cual básicamente se encarga de realizar lo que queríamos.

  Pygettext se encarga de coger todos los textos que quieras y generar un fichero con extensión .pot. Con este fichero .pot deberemos generar las distintas traducciones en los idiomas que queramos, para después dentro de nuestro software pygettext cargue la traducción que deseemos [15].

- **Diseños del Bot:** la fase de diseño de cualquier proyecto software es tan importante como la implementación del código. Por ello, había que buscar el mejor programa que satisficiera las distintas necesidades para crear diagramas UML.

[14] Kenneth Reitz (2013) Requests: HTTP para Humanos - http://es.python-requests.org/es/latest/
[15] Internacionalización de programas Python (2017) - https://www.freakspot.net/tag/pygettext/

Tras realizar un estudio sobre los distintos programas, llegué a la conclusión de que para realizar los distintos diagramas previos y posteriores a la implementación del Bot tenía que recurrir al programa Visual Paradigm. Visual Paradigm es uno de los softwares para crear los diagramas que componen el diseño de un software más completos del mercado, con hasta 13 tipos distintos de diagramas UML. [16]

2.6 Arquitectura software

Vamos a pasar a definir la arquitectura de nuestro Bot. A diferencia de la mayoría de los proyectos software, este Trabajo fin de Grado no se estructura en clases, sino en módulos. Esto tipo de estructura es un caso excepcional que se da en el lenguaje de programación Python, ya que permite hacer módulos en lugar de clases. De esta forma, decidí que quedaría mejor estructurado en módulos que representaran una tarea a realizar por el Bot. Tras todo lo anteriormente dicho, la estructura del Bot queda como sigue (Figura 2.2).

Como podemos observar en la imagen, cada módulo se compone de una serie de funciones encargadas de realizar todas las tareas que componen la función que tiene asignado cada uno de los módulos. Al no estar estructurado en clases, este Bot no dispone de una función main que se ejecute cuando arranca como ocurre en la inmensa mayoría de aplicaciones. Vamos a pasar a comentar cada uno de los módulos que componen la arquitectura del Bot:

- **Handler:** este módulo es el que hace de main, es decir, el que se ejecuta al arrancar el Bot. Como su nombre indica, este módulo se compone de un conjunto de handlers los cuales filtran los mensajes que va recibiendo el Bot y así llamar al módulo indicado. Estos handlers están configurados para recibir los siguientes comandos/mensajes:

[16] Visual Paradigm for UML - https://en.wikipedia.org/wiki/Visual_Paradigm_for_UML

- **Comando /start:** este comando sirve para iniciar la conversación con el Bot (llamar al módulo Start desde la función saludar del módulo Handler).

- **Comando /help:** este comando sirve para pedir al Bot que nos muestre una guía con todos los comandos (llamar al módulo Ayuda desde la función mostrarAyuda del módulo Handler).

- **Comando /tutorial:** con este comando indicamos al Bot que queremos que nos muestre un tutorial sobre como jugar a las Aventuras Conversacionales (llamar al módulo Tutorial desde la función mostrarTutorial del módulo Handler).

- **Comando /language:** este comando sirve para decirle al Bot que queremos cambiar el idioma (llamar al módulo language desde la función idioma del módulo Handler)

- **Comando /play:** este comando indica al Bot que queremos jugar a un nuevo juego (llamar al módulo Play desde la función jugar del módulo Handler).

- **Comando /about:** este comando sirve para pedirle al Bot que nos muestre información acerca de los desarrolladores del proyecto (llamar al módulo About desde la función mostrarAbout del Handler).

- **Otro mensaje:** si recibe otro mensaje que no sea uno de los comandos anteriores el Bot lo tomará como una acción de un juego y, por lo tanto, lo procesará como tal (llamar al módulo Play desde la función movimiento del módulo Handler).

Como se puede observar, cada una de las funciones del módulo Handler tiene dos versiones, una para Chats privados y otra para Grupos, ya que en función de uno u otro debemos manipular unos datos distintos en la Base de Datos.

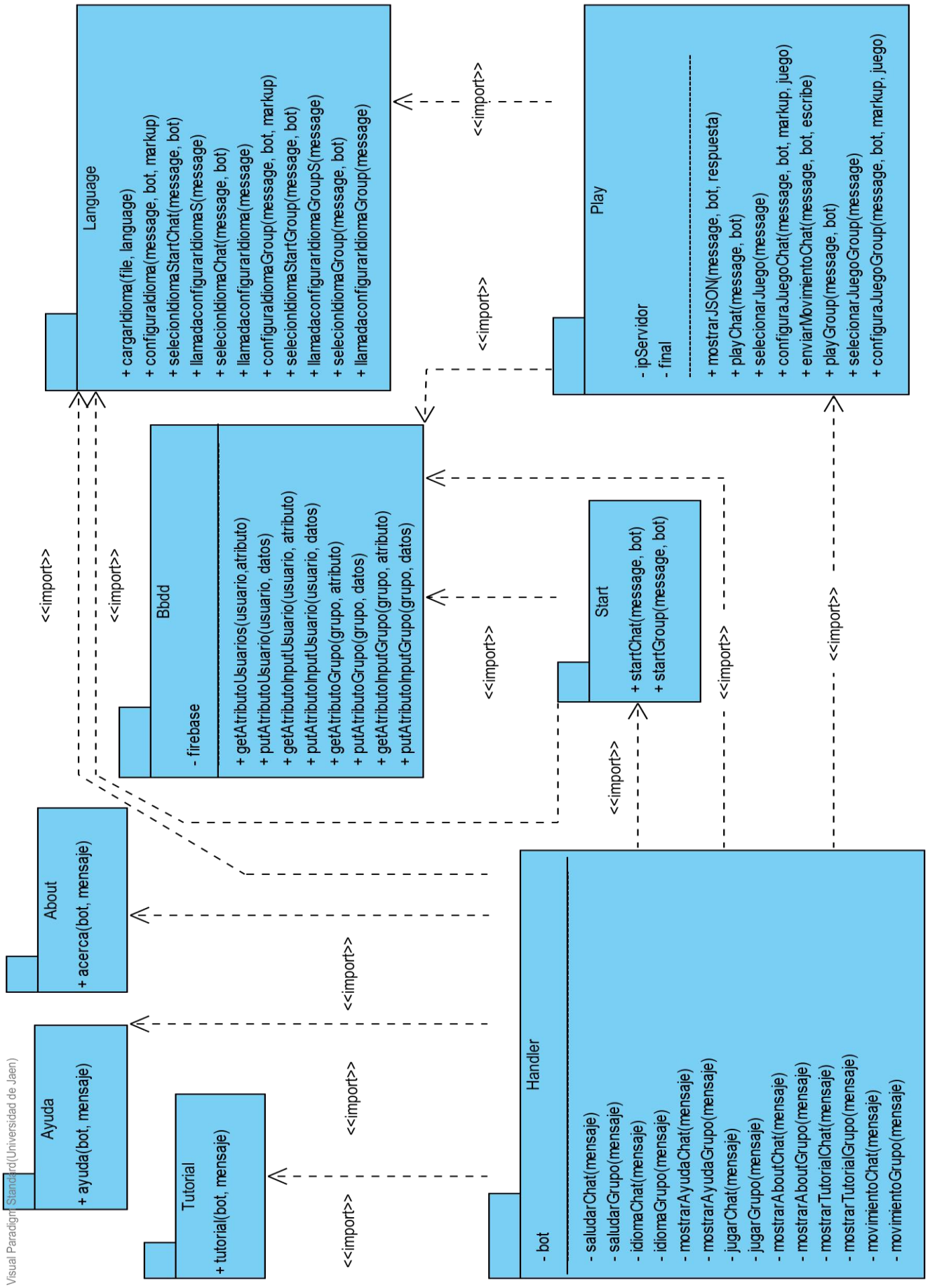


Figura 2.2 Diagramas de paquetes representando la arquitectura del Bot

- **Tutorial:** este módulo simplemente se compone de una función la cual se encarga de mostrar por el Chat un tutorial sobre cómo se juega a una Aventura Conversacional con el Bot.

- **Ayuda:** al igual que el anterior, este módulo simplemente se compone de una función encargada de mostrar por el Chat la lista con todos los comandos que tiene el Bot junto a una pequeña explicación de para qué sirve cada uno.

- **About:** al igual que los dos anteriores, este módulo solo se compone de una función encargada de mostrar por el Chat información relativa a cada uno de los desarrolladores del proyecto.

- **Start:** este módulo es el encargado de realizar la configuración inicial de cada grupo o usuario, es decir, registrar sus datos en nuestra Base de Datos. Además de esto, como se ha comentado anteriormente, envía un mensaje por el Chat saludando e indicando los pasos iniciales que debe tomar para aprender a usar el Bot.

- **Idioma:** este módulo es el encargado de cambiar el idioma cada vez que se lo pidamos. Para ello, dispone de una serie de funciones cuyo funcionamiento es el de crear un menú con los idiomas que podemos elegir y esperar a que el usuario o grupo elija el idioma deseado. Una vez elegido, el módulo hace los correspondientes cambios en la Base de Datos para que quede reflejado el nuevo idioma.

- **Bbdd:** este módulo es el encargado de realizar la conexión con la Base de Datos, insertando o extrayendo la información que el Bot solicite a través de cada una de las funciones que este módulo posee específicas en función de si la información a manipular es de un usuario o de un grupo. La arquitectura de la Base de Datos la explicaremos en detalle más adelante.

- **Play:** este es el módulo más complejo de la arquitectura del Bot, ya que abarca todo lo relacionado con la conexión con el Servidor para jugar las distintas Aventuras Conversacionales. Por ello, voy a explicar este módulo más a fondo:

  En primer lugar, tenemos la función mostrarJSON. Como su nombre indica, esta función se encarga de extraer el texto de los distintos JSON que recibe por parte del Servidor y mostrarlo por el Chat correspondiente.

  Después tenemos las funciones encargadas de comenzar un nuevo juego (funciones play, seleccionarJuego y configurarJuego). El procedimiento es el siguiente: en primer lugar, el Bot manda una petición al Servidor pidiendo la lista con todos los juegos disponibles. Una vez recibe la lista con todos los juegos, crea un menú (al igual que hacía para la selección del idioma) con los diferentes juegos, y espera la respuesta por parte del usuario/grupo.

  Una vez recibe el nombre del juego por parte del usuario/grupo, preparamos el JSON inicial para enviarle al Servidor indicando que queremos iniciar ese juego en cuestión. Después de esto, el Bot recibirá por parte del Servidor el primer JSON con el texto inicial del juego, mostrándolo por el Chat correspondiente.

  Luego tenemos la función encargada de enviar movimiento, el funcionamiento es parecido a lo anterior explicado, construimos un JSON, pero esta vez con el texto de la acción que el usuario o grupo ha enviado al Bot, y lo enviamos al Servidor. Una vez el Bot recibe la respuesta con la actualización del progreso del juego, muestra por el Chat correspondiente el contenido de la respuesta del servidor (el nuevo lugar o la nueva situación en la que se encuentre el jugador).

Una vez entrado en detalle en cada uno de los módulos que componen la arquitectura del Bot, vamos a especificar como sería el flujo de ejecución del Bot por parte de un usuario o grupo que comienza a interactuar con el:

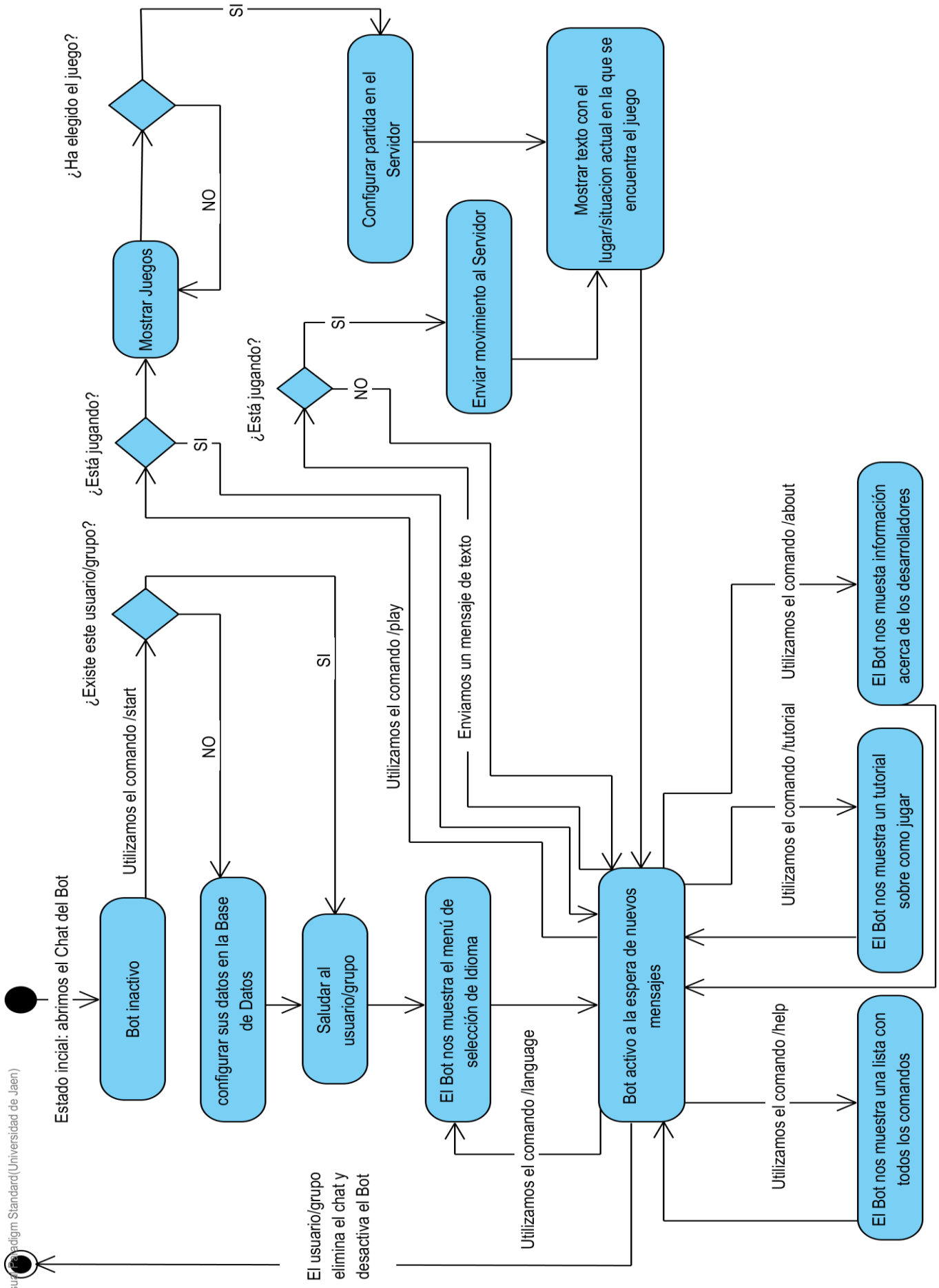


Figura 2.3 Diagrama de flujo de la ejecución del Bot con un usuario / grupo

Habiendo detallado toda la arquitectura del Bot, es hora de pasar a explicar cómo está estructurada la Base de Datos del mismo. Como se comentó anteriormente, esta Base de Datos está estructurada en un único JSON, por lo que no habría ningún diagrama entidad-relación, ya que toda la información va correlativa. Vamos a pasar a explicar cada uno de las variables y los atributos que se almacenan en el JSON de nuestra base de datos:

- **Users:** esta variable almacena una lista con todos los usuarios por su identificador único de Telegram. Cada usuario a su vez tendrá una seria de atributos que comentaremos más adelante.

- **Groups:** al igual que con los usuarios, esta variable almacena una lista con todos los grupos por su identificador único de Telegram. Cada grupo a su vez tendrá una serie de atributos que comentaremos más adelante.

- **FirstName:** variable que almacena el nombre de un usuario.

- **LastName:** variable que almacena el apellido del usuario (si tiene).

- **Nick:** variable que almacena el Nick del usuario.

- **Title:** variable que almacena el nombre completo de un grupo.

- **Playing:** variable que almacena si el usuario o grupo está jugando como si de una variable booleana se tratase (True/False).

- **Choosing:** variable que almacena si un usuario o grupo se encuentra escogiendo una opción de algún menú (idioma o elección de juego). Al igual que la variable anterior, esta actúa como un booleano (True/False).

- **Game:** variable que almacena el nombre del juego al cual está jugando el usuario o grupo.

- **Language:** variable que almacena el código regional correspondiente al idioma que el usuario haya elegido (“es”, “en”, etc.).

- **GameInput:** esta variable almacena, para cada usuario o grupo, una serie de variables que sirven para identificar nuestra partida entre todas las que puede haber simultáneamente ejecutándose dentro del Servidor. Cada una de las variables que incluye GameInput son las usadas para construir el JSON que se le manda al Servidor junto a nuestro movimiento para continuar jugando. Vamos a ver cuáles son estas variables:

    - **gen:** esta variable indica el número de movimientos que llevamos dentro de un juego.

    - **id:** variable con un número identificador único dentro del Servidor. Esta id cambia cada vez que cambiamos de juego.

    - **type:** variable que indica el tipo de texto que hemos recibido por parte del Servidor. Esta variable puede tener dos tipos de texto: “line” o “char”.

    Estas 3 variables explicadas funcionan en conjunto como una clave única para indicar al Servidor que nos devuelva nuestra partida y no la de otro usuario o grupo.

3. Puesta en marcha de la solución propuesta

Tras haber definido en detalle cómo ha sido llevado a cabo el desarrollo de la solución, es el momento de poner en marcha la solución y realizar las diferentes pruebas durante la implementación del Bot y tras la finalización del desarrollo de este.

3.1 Validación del proyecto

Durante la implementación del Bot, se ha realizado hincapié sobre todo en la conexión Bot – Servidor, ya que esto ha sido lo que más problemas ha dado a la hora de realizar las pruebas de ejecución del Bot, puesto que este dependía del Servidor que como se dijo es el encargado de ejecutar los juegos.

La principal causa de este problema recaía en que el servidor no devolvía donde se producía el error, si no que devolvía el código HTML 500. Este error nos indica que ocurrió un error interno en el Servidor. Básicamente, algo salió mal, pero el servidor no puede ser más específico sobre la condición del error en su respuesta al Bot. [17]

Arreglar este error fue una de las cosas que más tiempo nos llevó del desarrollo, ya que arreglar un error sin saber realmente su causa no es algo sencillo. Finalmente, tras ir realizando distintas pruebas de conexión con diferentes JSON y de diferentes formas, descubrimos cual era el error, el cual solucionarlo fue sencillo, permitiéndonos continuar con el desarrollo del Bot.

El error consistía en que el Bot no estaba mandando el JSON como el Servidor esperaba, por lo que no podía procesar la información que le llegaba para configurar la partida y, por lo tanto, le respondía con un código HTML 500 indicando que algo malo estaba ocurriendo.

[17] Error HTTP 500 Internal server error - http://www.checkupdown.com/status/E500_es.html

Otro problema que tuve durante la implementación del Bot fue la adaptación de este al multijugador, es decir, que se pudiera jugar desde un grupo. En un principio tenía pensado agregar un comando para activar/desactivar el Bot para que no estuviera pendiente de lo que se escribiera por un grupo.

De esta forma, la idea era que los integrantes de un grupo que estuvieran jugando una partida pudieran hablar para comentar que hacer en el siguiente paso sin que todo ese texto lo tomara el Bot como acciones para enviar al Servidor.

Sin embargo, vi que para que el Bot tomara el texto de un grupo como acción, este tenía que ser enviado respondiendo al Bot, y no simplemente escribiendo como se hacía en los chats privados. De esta forma, los integrantes de un grupo podían hablar sin que el Bot tomara todos los mensajes como una acción y, una vez el grupo tenga decidido que acción realizar, uno de los integrantes le debe comunicar al Bot la acción que desean hacer enviándosela como respuesta a uno de los mensajes del Bot.

Por lo tanto, finalmente no tuve que recurrir a la implementación de un nuevo comando para activar/desactivar el Bot en grupos como en un principio tenía pensado.

3.1.1 Prueba de la solución

Una vez testeado y comprobado que todo funciona correctamente, es hora de realizar las pruebas y evaluaciones definitivas de cara al público. Para ello, escogimos un conjunto de Aventuras Conversacionales tanto en inglés como en español, y pedimos a nuestra gente más cercana que pudiera usar el Bot que lo probara y posteriormente nos contara su experiencia con él.

Esta prueba se llevó a cabo durante un total de 2 días, en los cuales teníamos abiertas tanto la consola del Bot como la del servidor para ver el uso que estaban haciendo del Bot. Observando estas consolas se podía observar que la gente no

tenía dificultad a la hora de hacer uso del Bot, ya que en cuestión de un par de minutos tras comenzar a usarlo ya estaban jugando a una Aventura Conversacional.

Otro aspecto importante que se puedo observar durante las pruebas es lo bien que se comportaba el Bot ante los fallos en el uso por parte de los usuarios.

Tras la finalización de las pruebas, le pedía a cada uno de los usuarios que me contara su experiencia con el Bot. Prácticamente todos me respondían lo mismo: el Bot les gustaba.

En cuanto a la usabilidad, los usuarios opinaban que el Bot era muy intuitivo de usar gracias a toda la ayuda inicial que el Bot ponía a su disposición. En cuanto a los juegos, hubo variedad de opiniones en cuanto a gustos por un juego u otro. Sin embargo, todos coincidían en una cosa: las Aventuras Conversacionales son difíciles de jugar.

Esto es debido a que cada juego es totalmente distinto del resto y, por lo tanto, el repertorio de palabras que acepta como acciones es distinto. Entonces es posible que en una situación del juego no sepas avanzar, ya que tú sabes lo que hay que hacer, pero no de qué forma hay que transmitírselo al Bot para que el juego lo de por bueno.

Por ejemplo, observar y mirar a priori podrían servir cuando te piden que veas algo dentro del juego. Sin embargo, es posible que un juego solo admita mirar mientras que otro observar. En conclusión, la curva de aprendizaje en la mayoría de las Aventuras Conversacionales es difícil.

Otro aspecto por el que se le preguntó a los usuarios fue por el tiempo de respuesta, ya que por primera vez el Bot estaba siendo usado por varios usuarios simultáneamente. Los usuarios no tuvieron quejas respecto a los tiempos de respuesta, ya que el tiempo que tardaba el Bot en mostrarles los avances de los distintos juegos eran razonables según cada uno de los usuarios.

En resumen, la idea que presentaba este Bot agradó a todos los usuarios que lo probaron, llegando algunos incluso a estar jugando durante horas a algunos de los juegos, lo que demostró que hoy en día este tipo de juegos pueden gustar a mucha gente que los desconocía.

3.2 Manual del Desarrollador

A continuación, vamos a detallar los pasos a seguir para realizar una correcta puesta en marcha del Bot. Lo primero de todo es cumplir todos los requisitos previos para poder ejecutar correctamente el Bot: tener instalado un sistema operativo Windows o Linux, junto con el lenguaje de programación Python y unas series de dependencias/módulos, las cuales son:

- La API **pyTelegramBotAPI**, ya explicada anteriormente.
- La API de **firebase** para Python, también explicada anteriormente.
- Los módulos **locale** y **gettext**, encargados de traducir.
- El módulo **requests**, explicado anteriormente.
- El módulo **json**, explicado anteriormente.
- Los módulos **Thread** y **Queue**, utilizados para gestionar hilos. Estos dos módulos suelen venir instalador por defecto junto a Python

Una vez se cumplan esos requisitos, podemos proceder a la instalación de nuestro Bot. Lo primero que debemos hacer es elegir la versión del Bot acorde a nuestro sistema operativo: Windows o Linux.

Una vez elegida la versión del Bot, nos encontraremos con los siguientes archivos y carpetas tal y como aparecen en la siguiente imagen:

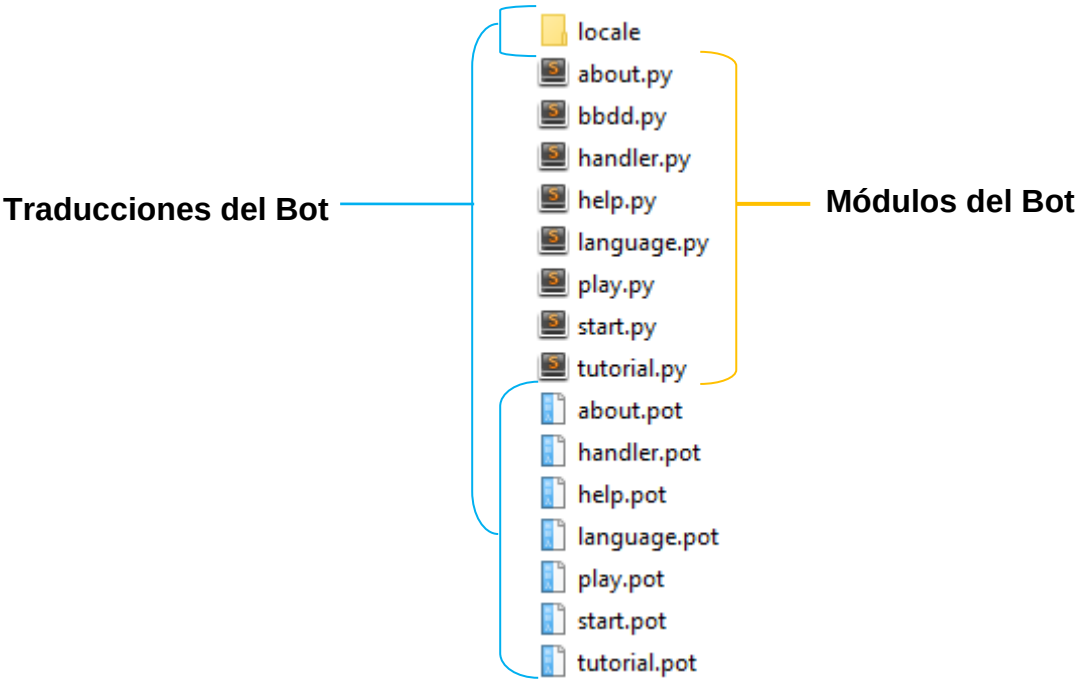


Figura 3.1 Organización de los distintos archivos del Bot

Como se puede observar en la imagen anterior, dentro de la carpeta se encuentran todos los módulos y traducciones del Bot. Antes de arrancar el Bot, deberemos configurarlo para que se conecte correctamente al Servidor. Para ello, debemos conocer la dirección IP del Servidor. Una vez la sepamos, abrimos el módulo **play.py** y editamos la variable **ipServidor** que encontraremos al comienzo del archivo con la IP del Servidor junto al puerto 4000, tal y como se puede observar en la siguiente imagen:

```
ipServidor = 'http://192.168.1.135:4000'
```

Figura 3.2 Ejemplo de configuración de la dirección IP del Servidor dentro del módulo play

Si estamos configurando el Bot en Linux, debemos realizar un paso extra antes de continuar. Tenemos que dirigirnos al módulo **language.py** y, dentro de este, dirigirnos a la primera función de todas **cargarIdioma**. Una vez situados en esta función, deberemos colocar la ruta al directorio **locale** que posee el Bot con las traducciones, tal y como se puedo observar anteriormente. A continuación, se puede

observar una imagen que deja más claro donde se ha de colocar la ruta al directorio **locale**.

```
def cargarIdioma(file, language):
        directorio = os.path.join(os.path.dirname(__file__), 'language.py')
        current_locale, encoding = locale.getdefaultlocale()
        print(directorio)
        idioma = gettext.translation(file, 'directorio locale', [language])
        idioma.install()
```

Figura 3.3 Configuración del módulo language (Solo en Linux)

Ahora vamos a pasar a la configuración de la Base de Datos (este paso es **opcional**). El Bot ya viene configurado para usar una base de datos Firebase creada por mí, a la cual solo yo tengo acceso con mi cuenta de Google. Para tener acceso a mi base de datos tendría que añadirle al proyecto. Otra opción que tiene es la de crear una nueva base de datos Firebase. Para ello, simplemente habría que entrar a firebase.google.com y crear un nuevo proyecto con el nombre que desee.

Una vez dentro de ese proyecto, dirígete al apartado Database y copia la URL que verá arriba, tal y como aparece en la siguiente imagen:

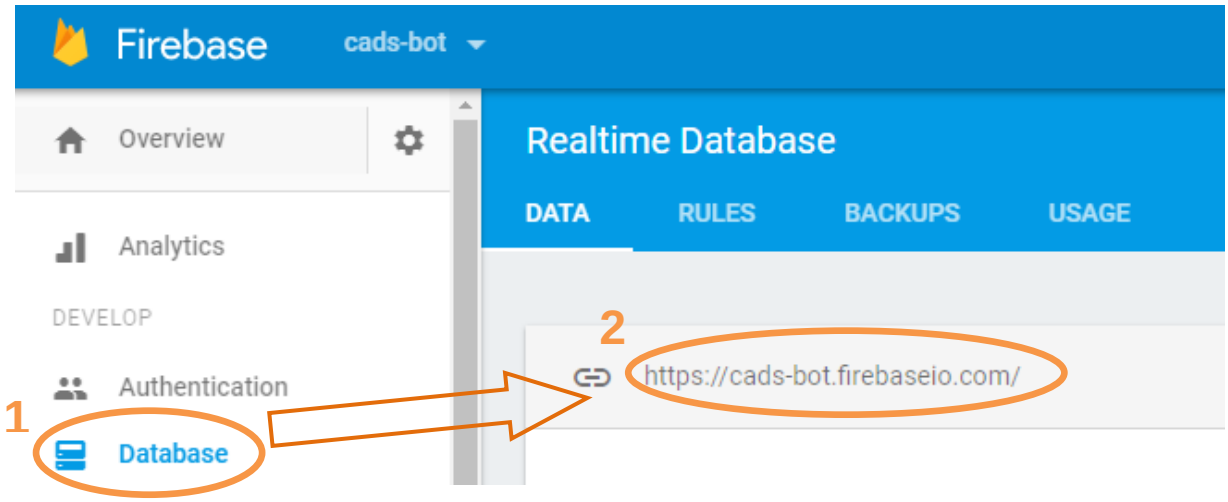


Figura 3.4 Configuración de Firebase

Cuando tengamos la URL de la base de datos, tenemos que dirigirnos al módulo **bbdd.py** y colocar la nueva URL, tal y como aparece en la siguiente imagen:

```
firebase = firebase.FirebaseApplication('https://cads-bot.firebaseio.com', None)
```

Figura 3.5 Configuración de la URL de la base de datos

Una vez realizado toda la configuración anterior, podemos dar paso a la ejecución del Bot. Para ello, basta con ejecutar el módulo **handler.py** con Python dentro de una consola de comandos CMD (si estamos en Windows) o Terminal (si estamos en Linux).

```
>python.exe handler.py
```

Figura 3.6 Ejemplo de ejecución del Bot en una consola de comandos CMD de Windows

Si todo se ha realizado correctamente, podríamos probar el Bot y recibir respuesta del mismo.

4. Conclusiones y trabajo futuro

Tras haber explicado en detalle cómo se compone el Bot y como se ha ido desarrollando cada una de las tareas que han contribuido a la realización de este Trabajo fin de Grado, es la hora de sacar las conclusiones de este trabajo.

Desde que conocí el mundo de Telegram y la gran variedad de Bots que este posee, siempre tuve la curiosidad por saber cómo se hace uno, y gracias a este Trabajo fin de Grado se me brindó la oportunidad de conocer más a fondo los Bots de Telegram y poder crear el mío propio. La funcionalidad de nuestro Bot es algo nunca visto entre toda la variedad de Bots de Telegram, cosa que añadió dificultad a este trabajo, pero que una vez superado, hace que la experiencia adquirida durante el transcurso de este trabajo haga que domine casi al completo la API de Telegram.

Además de esto, hay que recordad que este Trabajo fin de Grado está dividido en dos partes, Bot y Servidor, lo que hace que tanto yo como mi compañero hayamos obtenido más experiencia en cuanto a trabajo en equipo, algo que nos vendrá bien para el futuro.

En cuanto al lenguaje de programación Python, he aprendido mucho sobre este lenguaje gracias a este trabajo, ya que mis conocimientos sobre este lenguaje antes de comenzar a realizar el trabajo eran muy básicos.

Finalmente, y para concluir este trabajo, se deja abierta la posibilidad como trabajo futuro el seguir mejorando el Bot, sobre todo en el aspecto de la experiencia del usuario, añadiendo ayudas para cada juego y que así el jugador sepa que puede hacer en caso de que se quede atascado en alguna parte de un juego.

Bibliografía

[1] Aventura conversacional - https://es.wikipedia.org/wiki/Aventura_conversacional

[2] Inform - https://en.wikipedia.org/wiki/Inform

[3] Telegram Messenger - https://es.wikipedia.org/wiki/Telegram_Messenger

[4] Telegram Bot API - https://es.wikipedia.org/wiki/Telegram_Bot_API

[5] MTProto - https://es.wikipedia.org/wiki/MTProto

[6] JSON - https://es.wikipedia.org/wiki/JSON

[7] Android 4.1 Jelly Bean y la accesibilidad (2013) - https://elandroidelibre.elespanol.com/2013/03/especial-android-y-las-aplicaciones-para-personas-ciegas.html

[8] Mario Pérez Esteso (2017) - Programación extrema: qué es y principios básicos - https://geekytheory.com/programacion-extrema-que-es-y-principios-basicos

[9] Python - https://es.wikipedia.org/wiki/Python

[10] Komodo IDE: The best IDE for web and mobile app development - https://www.activestate.com/komodo-ide

[11] Los 10 mejores IDE´s para Python (2016) - http://notasinformaticas.com/los-10-mejores-ides-para-python/

[12] Firebase - https://en.wikipedia.org/wiki/Firebase

[13] eternnoir (2015) pyTelegramBotAPI - https://github.com/eternnoir/pyTelegramBotAPI

[14] Kenneth Reitz (2013) Requests: HTTP para Humanos - http://es.python-requests.org/es/latest/

[15] Internacionalización de programas Python (2017) - https://www.freakspot.net/tag/pygettext/

[16] Visual Paradigm for UML - https://en.wikipedia.org/wiki/Visual_Paradigm_for_UML

[17] Error HTTP 500 Internal server error - http://www.checkupdown.com/status/E500_es.html