UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Implementación del juego de las damas en un robot NAO

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Zayda Ferrer Lluch

Tutores: Vicente Javier Julián Inglada
Carlos Carrascosa Casamayor

2015 - 2016

Resumen

Este proyecto presenta una aplicación en donde un robot humanoide NAO juega a las damas contra una persona u otro robot en un ámbito doméstico. Nuestro robot NAO piensa y calcula las jugadas dando constancia verbalmente a su adversario del movimiento que desea realizar, y captura las jugadas de su adversario a través del reconocimiento de voz.

El control global del sistema muestra cómo se han integrado diferentes tecnologías empleadas como la robótica y la inteligencia artificial.

A lo largo del texto se presentan tanto aspectos técnicos como pedagógicos, así como los elementos tecnológicos utilizados y la explicación de las técnicas utilizadas.

Para el desarrollo del proyecto se ha utilizado el software de programación y simulación Choregraphe y NAOqi, el lenguaje de programación Python y el robot NAO, en el que se ha comprobado el correcto funcionamiento del sistema desarrollado.

Tanto al final del proyecto, como a lo largo del desarrollo de éste, se han realizado diversas pruebas para validar el funcionamiento del sistema desarrollado. En dichas pruebas se comprobó por una parte la eficiencia y el tiempo de cómputo de algoritmo, y por otra parte la correcta integración y funcionamiento en un robot NAO tanto en un espacio simulado como en un espacio real.

Palabras clave: Robótica, Inteligencia Artificial, Robots NAO, Reconocimiento del habla, juego de las damas.

Abstract

This Project presents an application where a humanoid robot NAO plays checkers against a person or another robot in a domestic environment. Our NAO robot thinks and calculates the moves giving to his opponent verbally record of the move to perform, and captures the moves of his opponent through voice recognition.

The overall control of the system displays how we integrated several technologies such as robotics and artificial intelligence.

Along the text both technical and pedagogical aspects are presented as well as the technological elements used and the explanation of the techniques used, are also presented.

To develop the Project has used the programming software and simulation Choregraphe and NAOqi, the programming language Python and the NAO robot, which has been proven the right operation of the developed system.

Both, at the end of the Project, and throughout the ddevelopment of this, several tests were performed to validate the operation of the developed system. In these tests, both, efficiency and the computaton time of the algorithm were tested, and also the correct integration and operation on a NAO robot were tested in as well in a virtual reality as in real space with a real NAO robot.

Keywords : Robotics, Artificial Intelligence, NAO Robots, Speech Recognition, Checkers.

Tabla de contenidos

1. Introducción 8
2. Objetivos 10
   2.1. Subobjetivos 10
3. Trabajo relacionado 11
   3.1. Robótica 11
      3.1.1. Robot NAO 15
   3.2. Juegos de mesa: 21
      3.2.1. Beneficios de los juegos de mesa: 21
      3.2.2. El juego de las Damas: 24
   3.3. Inteligencia Artificial (IA) y los juegos de mesa 26
      3.3.1. Aplicaciones de la IA 26
      3.3.2. ¿Por qué es importante la robótica y la IA? 27
      3.3.3. Algoritmos para juegos 27
   3.4. Python 34
4. Trabajo realizado 35
   4.1. Planteamiento del proyecto 35
      4.1.1. Toma de decisiones 35
      4.1.2. Las damas españolas 35
      4.1.3. Programación con Python 35
   4.2. Desarrollo del programa principal para jugar a las damas 36
   4.3. Algoritmo de búsqueda con adversario, Minimax 38
      4.3.1. Max - Nivel 1: 38
      4.3.2. Min - Nivel 2: 39
      4.3.3. Calculo de la función de utilidad: 41
      4.3.4. Poda Alfa-Beta: 42
   4.4. Adaptación del algoritmo desarrollado al middleware NAOqi 42
      4.4.1. ALMemory 45
      4.4.2. ALTextToSpeech 46
      4.4.3. ALSpeechRecognition 47
      4.4.4. ALLeds 48
   4.5. Utilización de Choregraphe 50

5. Validación 53
5.1. Algoritmo 53
5.1.2. Pruebas de eficiencia y tiempo de cómputo 55
5.2. Simulación 56
5.3. Pruebas físicas 58
6. Conclusiones 60
7. Bibliografía 63
8. Apéndice A 64
A.1. Python 64
A.3. Choregraphe 64
A.2. NAOqi 64
9. Apéndice B 66
Agradecimientos: 71

1. Introducción

Recientemente se ha hablado mucho sobre la robótica y los robots y sobre el uso que se le puede dar a éstos. Muchos han sido los avances en este sector y en el sector de la inteligencia artificial, pero aun así, muchos son los avances que están por llegar. Si bien es cierto que la curiosidad no es un rasgo característico o único de las personas, pues muchos otros animales también lo tienen, es cierto que otras características como el razonamiento o el aprendizaje nos han permitido poder investigar, logrando avances que mejoran y/o facilitan nuestras vidas. Y es precisamente esta curiosidad la que genera un ansia de seguir investigando, descubriendo, creando, etc… Y es en este momento, cuando nos centramos entre otros en el campo de la inteligencia artificial y la robótica.

Por otra parte, a lo largo de muchísimos años también se ha hablado mucho sobre los beneficios de los juegos de mesa tanto en niños como en adultos. Y es justo en esta intersección cuando nos planteamos el objetivo del presente el proyecto.

Dicho proyecto consiste en incorporar los avances en robótica y en inteligencia artificial en el día a día de las personas en general, para de esta forma poder proporcionar estos avances tecnológicos con sus correspondientes beneficios en el ámbito doméstico.

Si bien es cierto que ya existen programas de ordenador que te permiten jugar cuando no tienes adversario, es decir, que te permiten jugar contra el ordenador, estos adversarios, siguen siendo pantallas. Y jugar contra pantallas es algo que si bien los jóvenes tienen bastante asimilado, y en su mayoría de hecho juegan a juegos virtuales, sino diariamente, al menos una vez semanalmente; para una gran mayoría de personas adultas sigue siendo una barrera. Y para poder tumbar esa barrera, acaso ¿no sería interesante poder jugar físicamente en la mesa de tu salón?

De esta forma conseguiríamos una interacción real, en este caso con un agente inteligente, que reaccione antes las diversas situaciones del juego.

Por tanto, ¿qué mejor que un robot humanoide para conseguir así un parecido más real a lo que sería jugar contra otra persona? Con todo esto, se planteó el desarrollar un sistema que permita jugar a un robot bípedo a un juego típico doméstico. En concreto se planteó implementar el algoritmo del juego de las damas sobre un robot bípedo NAO.

Finalmente, partiendo que “Real-Time Artificial Intelligence” es una disciplina que incorpora las técnicas de resolución de problemas usadas en inteligencia artificial con las restricciones de tiempo real, sabiendo que estas situaciones necesitan de respuestas válidas en intervalos de tiempos acotados para garantizar el correcto funcionamiento del sistema, se crea entonces, la necesidad de adaptar las técnicas utilizadas en inteligencia artificial para poder aplicarlas en estos entornos reales. Y sabiendo también el objetivo del proyecto, se puede ver perfectamente la necesidad de adaptar ciertas

técnicas de la inteligencia artificial como puedan ser los algoritmos de búsqueda, a un entorno real, como también lo es un entorno doméstico. Con todo esto, el objetivo del proyecto se enmarcaría en la línea de investigación de “Real-Time Artificial Intelligence” del departamento del GTI – IA.

2. Objetivos

El objetivo principal del proyecto consiste en desarrollar un algoritmo que permita a un robot NAO jugar a las damas, así como un mecanismo adecuado de interacción con el jugador humano.

Este proyecto a su vez, contiene una serie de subobjetivos implícitos que se describen a continuación.

2.1. Subobjetivos

- Investigar y analizar entre las distintas variedades de juegos de las damas.
- Investigar y analizar los distintos algoritmos de búsqueda de juegos con adversario.
- Aprendizaje del software de desarrollo para los robots NAO.
- Diseño e implementación de los algoritmos seleccionados sobre la plataforma de programación de los robots NAO.
- Integración del algoritmo principal desarrollado en Python en el software de desarrollo de los robots NAO.
- Desarrollo de diferentes pruebas para evaluar el funcionamiento del software desarrollado.

3. Trabajo relacionado

Antes de seguir avanzando en el tema, hay una serie de conceptos que deberíamos tener claros. Tanto conceptos generales como por ejemplo … ¿qué es la robótica?, ¿qué son los robots?, ¿qué es la inteligencia artificial?, ¿para qué se pueden utilizar los robots?, ¿cuáles son los diferentes juegos de mesa y en que consisten?, ¿cuáles son los beneficios?, etc… Como otros detalles o conceptos más técnicos como por ejemplo ¿qué son los robots NAO?, ¿cuáles son sus especificaciones técnicas? O ¿cómo se programan?, ¿cuáles son los distintos algoritmos de búsqueda que podemos utilizar para implementar el juego de las damas?, etc…

Profundicemos entonces, más técnicamente en estos conceptos.

3.1. Robótica

Según la Real Academia de la lengua Española (RAE), un robot es "*Una máquina o ingenio electrónico programable, capaz de manipular objetivos y realizar operaciones antes reservadas sólo a las personas*".

En otras palabras y profundizando un poco más en el concepto de robot, los robots son máquinas o agentes físicos que pueden ser utilizados para realizar trabajo. Algunos robots pueden trabajar por sí mismos y otros necesitan tener constantemente a una persona cerca diciéndoles que es lo que deben hacer.

Los componentes básicos de un robot son: el cuerpo, el sistema de control, elementos de manipulación y elementos para el desplazamiento (aunque no siempre).

Los robots se pueden clasificar según su cronología en diferentes generaciones, o según su arquitectura en:

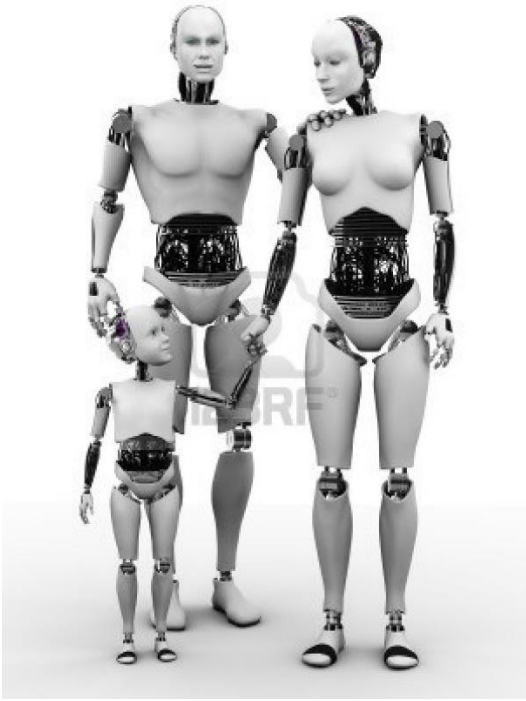
- Poliarticulados: Generalmente estáticos y con pocos grados de libertad, como por ejemplo los manipuladores, robots industriales, robots cartesianos, etc…

Robot poliarticulado

- Móviles: Con capacidad de desplazamiento, los hay tanto autómatas como guiados (con telemando).

Robot móvil

- Androides: Aquellos que intentan reproducir total o parcialmente la forma y comportamiento del ser humano.



Familia de Robots Androides

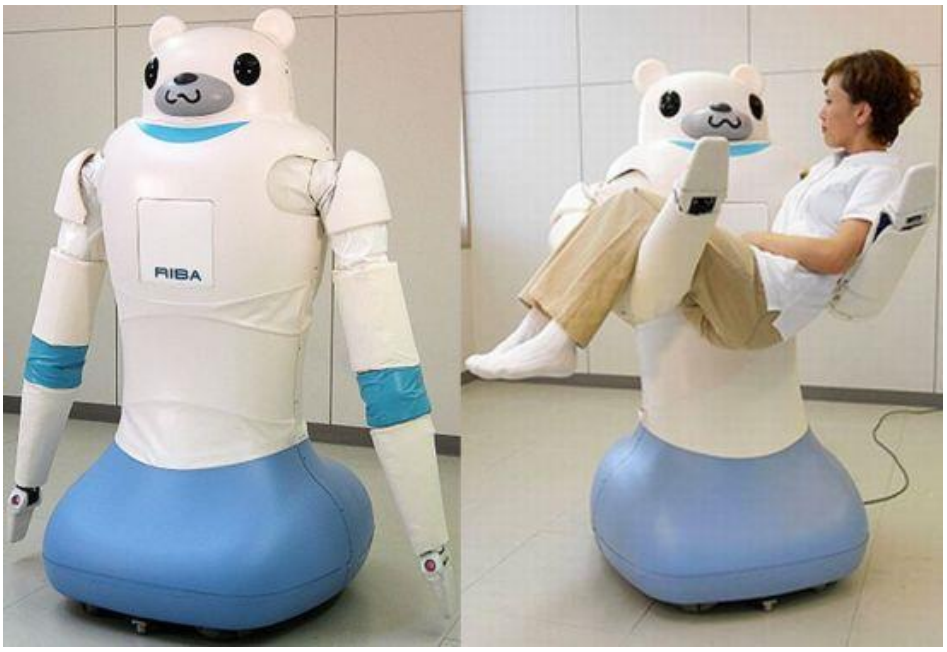
Robot androide bípedo NAO

- Zoomórficos: Estos también podrían incluir a los androides ya que se trata de robots que imitan los sistemas de locomoción de diversos seres vivos. Generalmente muy utilizados también para la exploración espacial.

Robot Zoomórfico perro

Robot zoomórfico serpiente

- Híbridos: Aquellos de difícil clasificación cuya estructura combina dos o más de las características descritas anteriormente.



Robot híbrido

Una vez explicado que son los robots, si seguimos sumergiéndonos en el mundo de la robótica y buscamos su definición en el diccionario de la Real Academia de la lengua Española, nos encontramos con que "*La robótica es una técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos*".

Por tanto, se puede deducir fácilmente que la robótica es la ciencia y la técnica que está involucrada en el diseño, la fabricación y la utilización de robots.

La informática, la electrónica, la mecánica y la ingeniería son sólo algunas de las disciplinas que se combinan en la robótica.

El escritor estadounidense Isaac Asimov (1920-1992), suele ser considerado como el responsable del concepto de la robótica. Además, en sus libros *Yo, robot* (1950) y *El segundo libro de robots* (1964), Isaac Asimov fijo lo que él llamó las tres leyes de la robótica, que son un conjunto de normas escritas que la mayoría de los robots de sus cuentos y novelas están diseñados para cumplir. Y aunque actualmente no están instauradas como tales en la robótica, en un futuro no muy lejano, de seguir con los avances tecnológicos, podríamos vernos obligados a instaurarlas oficialmente como leyes de la robótica. Las tres leyes de Isaac Asimov son:

1. Un robot no hará daño a un ser humano o, por inanición, no permitirá que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la primera ley.

3. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o la segunda ley.

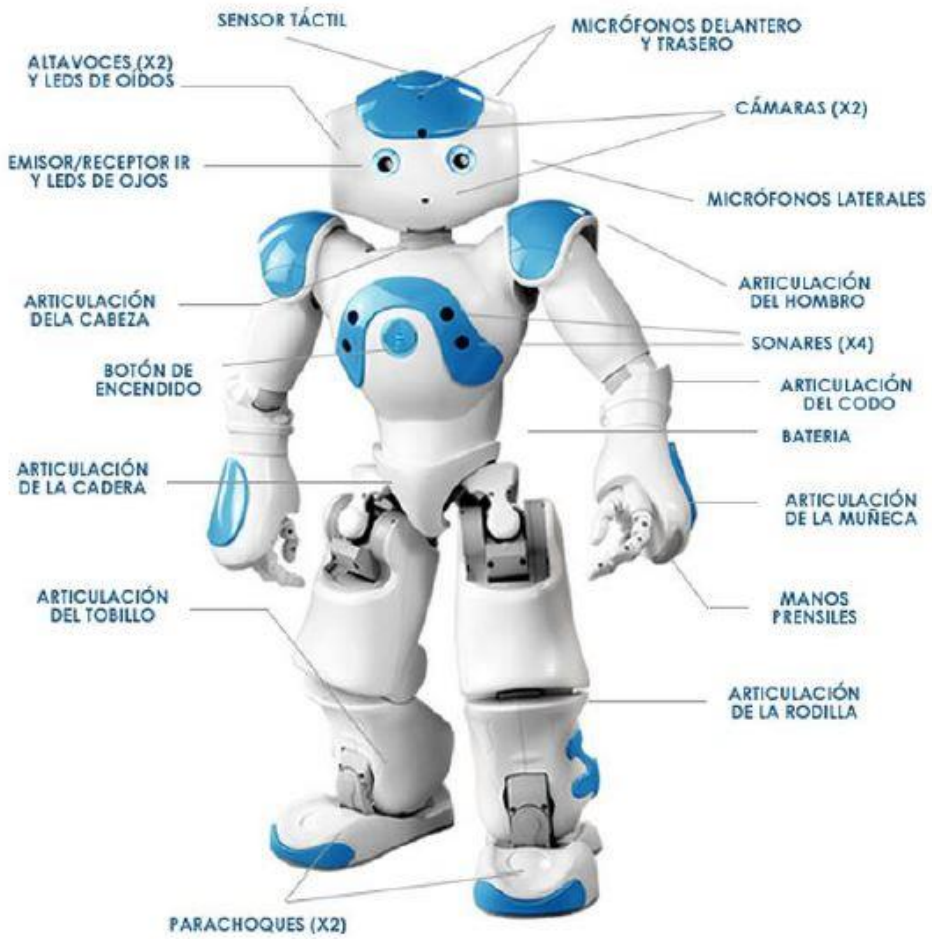
Los avances en la robótica han dado lugar al desarrollo de una serie de disciplinas relacionadas. Tres son las más grandes categorías en las que podemos dividir el uso de la robótica en la actualidad:

1. *Robótica en la industria:*
   Existen numerosos trabajos que requieren un alto grado de precisión y velocidad, como por ejemplo embalaje, ensamblado, pintura, etc...
   La Federación internacional de Robótica calculó que hay más de 1.3 billones de robots industriales operando alrededor del mundo.
2. *Robótica en la investigación:*
   Son robots utilizados para desarrollar tareas peligrosas o para acceder a lugares de difícil acceso. El mayor ejemplo de la utilización de robots en la investigación lo podemos ver en las agencias espaciales como por ejemplo "La Nasa".
3. *Robótica en la educación:*
   La robótica se ha convertido en una herramienta accesible e interesante para educar y apoyar campos como la ciencia, la tecnología, la ingeniería, las matemáticas, etc...
   La robótica permite a los estudiantes usar sus mentes y sus manos para crear como ingenieros, artistas, técnicos, o como si de todo a la vez se tratara.
   En el sistema educativo de hoy en día con sus limitaciones presupuestarias, las escuelas e institutos están en una constante búsqueda de maneras interesantes y rentables de conseguir programas de alto impacto que integren tecnología con múltiples disciplinas, y por ello, hoy más que nunca, están adoptando la robótica en el aula para cumplir con mayores niveles académicos.

3.1.1. Robot NAO

NAO es una línea de robots programables y autónomos puestos a la venta por una empresa francesa conocida como Aldebaran Robotic, perteneciente al grupo SoftBank, en el año 2008.

Los robots NAO tienen forma humanoide, y cuentan con un sistema multimedia a bordo basado en Linux llamado NAOqi que hace uso de los recursos disponibles en los robots NAO, incluyendo dos cámaras HD, cuatro micrófonos, un sonar, dos emisores y receptores de infrarrojos, una unidad de medición inercial, nueve sensores táctiles y ocho sensores de presión. También cuenta con Wi-Fi y conexión Ethernet, Un conector USB que nos permite conectar Kinect o el sensor 3D Asus o un dispositivo Arduino. Funcionan con batería. Y tienen 25 grados de libertad, por lo que poseen una amplia movilidad.



Características de los robots NAO

Las características más importantes se muestran en la siguiente tabla.

Altura	58 cm
Peso	4.3 kg
Energía	Batería Li-Ion, 6 celdas en seria, Vnom = 21.6 V, I = 2 A
Autonomía	60 minutos (uso activo), 90 minutos (uso normal)
Grados de libertad	Entre 21 y 25 grados
CPU	Intel Atom @ 1.6 GHz

Desarrollado en el SO	NAOqi (Basado en Linux)
SO compatibles	Windows, Mac OS, Linux
Lenguaje de programación	C++, Python, Java, MATLAB, Urbi, C, .NET
Visión	Dos cámaras 1280x960 HD
Conectividad	Ethernet, Wi-Fi
Sensores	Giroscopio, Acelerómetro, Bumpers, Sonar, I/R

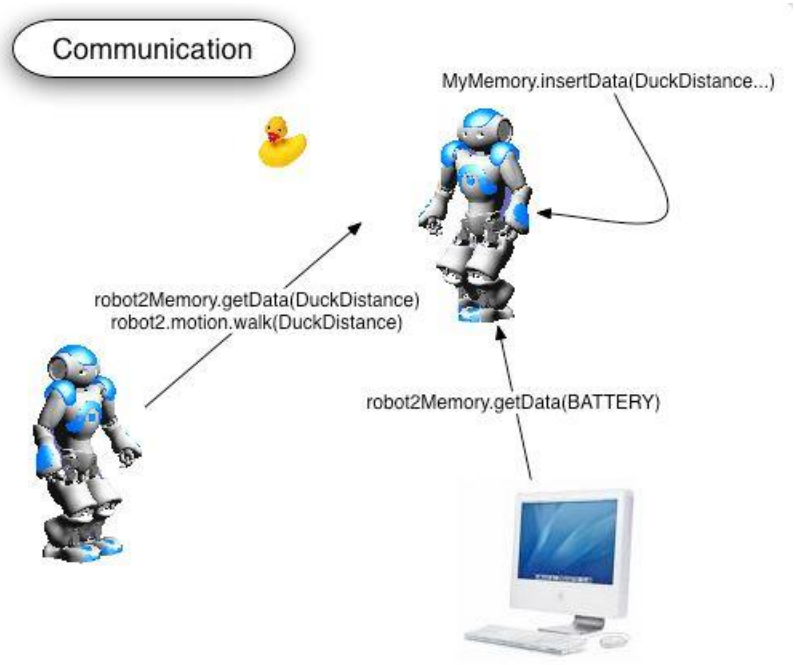
[Tabla de características]

Los robots NAO vienen con un paquete de software que incluye una herramienta grafica de programación ("Aldebaran Choregraphe"), un software de simulación y un kit de desarrollo software que describiremos a continuación.

3.1.1.1. NAOqi OS:

NAOqi es el Framework desarrollado por Aldebaran y usado por los robots NAO, responde a las necesidades básicas de los robots NAO como es el paralelismo, la utilización de recursos, la sincronización, eventos, etc...

Este framework permite una comunicación homogénea entre los diferentes módulos (locomotor, de audio, de video), y también la programación y la compartición de recursos de manera homogénea.



Conexión con los robot NAO – NAOqi

El robot NAO puede ejecutar un programa/algoritmo almacenado en su propia memoria interna, o se puede conectar a un dispositivo externo (un ordenador) a través de internet (tanto con Ethernet como a través de Wi-Fi), en cuyo caso, el programa puede estar almacenado en esta memoria remota y ser ejecutado por el robot sin necesidad de almacenarlo en la propia memoria del robot NAO.

El framework NAOqi permite acceder al robot a través de una conexión Wi-Fi para darle órdenes sin necesidad de cargar los códigos en la propia memoria del robot. Por tanto el framework sirve de puente entre la computadora y el robot NAO, controlando la ejecución de cualquier comportamiento y capturando la información de todos los sensores con el único objetivo de que todo se produzca con éxito y sin incidentes.

El framework se encuentra tanto dentro del robot, a modo de cerebro, como también en el ordenador desde el que se envían las ordenes.

Está dividido en seis grandes áreas claramente diferenciadas por su funcionalidad:

1. ***NAOqi Core***: Contiene un conjunto de módulos relacionados con los comportamientos genéricos del robot, la memoria del robot y los ficheros de configuración.
2. ***NAOqi Motion***: Contiene funciones que permiten al robot desplazarse, controlando el equilibrio, el estado de las articulaciones y la prevención de choques o caídas.

3. ***NAOqi Audio***: Contiene los módulos que le permiten al robot interactuar con el usuario mediante el habla, la reproducción de sonidos y el reconocimiento de voz.
4. ***NAOqi Vision***: Contiene los módulos que permiten grabar desde las cámaras del robot NAO y realizar el reconocimiento de caras y objetos.
5. ***NAOqi Sensors***: Contiene los módulos que proporcionan información sobre el estado del robot: sensores, batería, pose en la que se encuentra, etc…
6. ***NAOqi Trackers***: Contiene las funciones que permiten seguir a una persona a través del reconocimiento facial, a un objeto o a una marca.

Para lograr una mayor flexibilidad y personalización, en lugar de utilizar Choregraphe, para programar nuevos módulos a través de la API de NAOqi se pueden emplear 9 lenguajes de programación distintos: Python, C++, .NET, C#, Visual Basic, F#, Java, Matlab o Urbi.

Aunque lo más común es utilizar librerías Python o C++, también es habitual entrar códigos que combinan el API de NAOqi con paquetes de ROS (Robot Operating System).

3.1.1.2. Simulador Choregraphe

Choregraphe es una aplicación multi-plataforma de escritorio con el fin de facilitar al desarrollador la programación de los robots NAO y que permite trabajar y/o probar los códigos tanto en un robot real como en un robot virtual.

Choregraphe permite diseñar y ejecutar distintos comportamientos, animaciones y creencias, (como la interacción con personas, bailar, enviar e-mails, etc…), acceder a los recursos del robot, monitorearlo y controlarlo, e incluso añadirle tu propio código "Python". Choregraphe es una herramienta fundamental para conocer el estado del robot y para poder trabajar independientemente con cada uno de sus recursos.

Todo lo que puedes hacer a través de NAOqi se puede hacer también a través de Choregraphe, ya que Choregraphe tiene acceso a toda la API de NAOqi, el único inconveniente, es que las creencias programadas en Choregraphe "podrían" ejecutarse más lentamente que aquellas programadas directamente en NAOqi. Por otra parte, Choregraphe facilita la interacción con el robot NAO ya que proporciona la opción de crear animaciones y programas rápidamente, en ocasiones incluso sin escribir una sola línea de código, tan solo gracias a la interfaz gráfica, sencilla e intuitiva de Choregraphe.

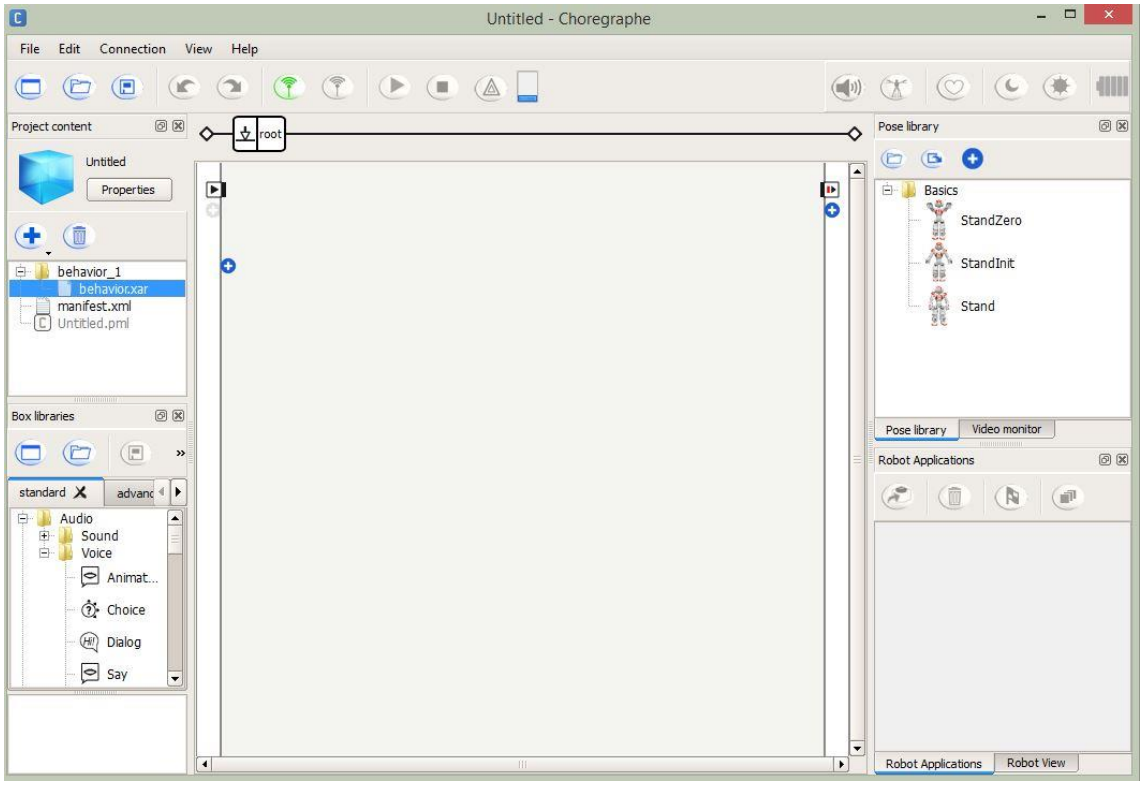
Pero no todo podía ser perfecto, ciertas funcionalidades no se pueden probar con Choregraphe conectado a un robot virtual, como por ejemplo la funcionalidad de "Speech Recognition" o reconocimiento de voz del robot,

aunque si se podrá probar si se conecta a un robot NAO real en lugar de al robot NAO virtual.

Utilizando Choregraphe…

Las diferentes secciones de la interfaz son generalmente las siguientes:

- Menú a la Izquierda: Contiene módulos con fragmentos de códigos para ejecutar directamente sobre el robot
- Menú a la derecha: Se puede visualizar el estado del robot (en la parte inferior), y controlar o cambiar la posición del robot (en la parte superior)
- Menú Superior: Contiene todas las demás opciones (conexión, importar/exportar proyecto, guardar, librerías, etc…)



Captura de pantalla al iniciar Choregraphe

Para que el robot del simulador realice un movimiento podemos combinar los comportamientos ya desarrollados del menú de la izquierda, o crear otros nuevos mediante la opción de “animación”. Para ejecutarlos, simplemente hay que pulsar el botón “Run”. Al trabajar siempre sobre diferentes ventanas todo está muy bien ordenado y la flexibilidad que ofrece es inmensa. A través del menú superior se puede acceder al resto de información que esta herramienta proporciona.

Aunque estas son las secciones por defecto de la interfaz, se pueden modificar, desplazarlas a otro lugar, agrandar o reducir, añadir otras secciones o eliminar alguna de las secciones iniciales hasta conseguir una interfaz a tu propio gusto o según tus necesidades.

3.2. Juegos de mesa:

Sin duda algo difícil de definir, y más actualmente. Pero podríamos decir que inicialmente, los juegos de mesa eran aquellos juegos que requerían de una mesa o un soporte similar para jugarse, y que se suelen jugar con un grupo de personas alrededor del juego (entendiendo por grupo de personas, de 2 a más personas).

Lo que sí se puede afirmar rotundamente es que los juegos de mesa son claramente de tipo social, cultural e incluso didácticos.

3.2.1. Beneficios de los juegos de mesa:

Los juegos de mesa no sólo sirven para distraer y divertir a los niños, además influyen de manera significativa en su aprendizaje y en su adaptación social.

Los juegos de mesa pueden ser muy educativos y ayudan a que los niños desarrollen sus capacidades motoras, mentales y sensoriales, manteniendo su mente activa.

Los juegos además les permiten conocerse mejor a sí mismos, ¿Son competitivos? ¿Tienen paciencia? ¿Son diplomáticos? ¿Se comunican bien? Etc…

El experto mundial en psicología evolutiva, el catedrático español y miembro del observatorio del juego infantil, José Luis Linaza, *[Ref_1]* destaca que los juegos de mesa contribuyen al desarrollo físico, cognitivo y social, y favorecen la comprensión y aceptación de las reglas y normas deportivas, fomentando la salud psicológica, la concentración, la capacidad de asociación y agilidad mental, la resolución de problemas, las habilidades sociales, la participación, la constancia, el juego en equipo y la autonomía entre otras muchas ventajas.

Hay muchos tipos de juegos de mesa para desarrollar diferentes capacidades:

- Juegos de estrategia: Juegos como el ajedrez o las damas enseñan a los niños a elaborar sencillas estrategias, marcar planes o tomar decisiones, así que son perfectos para estimular la autonomía de cualquier niño.
- Juegos de concentración: Perfectos para mejorar la memoria y la capacidad de concentración de los niños.
- Juegos educativos: Les permiten aprender vocabulario, geografía, historia, etc… dependiendo del juego. O incluso les puede ayudar a reforzar su creatividad.

3.2.1.1. Beneficios de los juegos de estrategia como las Damas:

Centrándonos en los juegos de estrategia, que son el tipo de juegos que nos incumben para esta ocasión, puesto que se va a desarrollar un algoritmo que permita a un robot NAO jugar al juego de las damas. Como se ha explicado anteriormente, los juegos de mesa suelen aportar gran cantidad de beneficios en los niños, y son los juegos de estrategia, los juegos que mayor cantidad y variedad de beneficios aportan tanto en el desarrollo de los más pequeños, como al desarrollo de los más adultos.

Buceando un poco por internet se pueden encontrar gran cantidad de beneficios y el porqué de estos beneficios, algunos de los cuales se explican a continuación:

- Nuevas conexiones neuronales: Jugar a juegos de estrategia hace crecer las dentritas, creando nuevas conexiones neuronales.
- Ejercitan ambos hemisferios cerebrales: Se necesita utilizar el hemisferio izquierdo para reconocer los objetos (o las piezas), y el hemisferio derecho ayuda a reconocer los patrones o jugadas a realizar.
- Prevenir el Alzheimer: Según estudios realizados en el Colegio de Medicina Albert Einstein, jugar a juegos de estrategia estimula la función cerebral, disminuyendo el riesgo de padecer demencia y combatiendo los síntomas de esta, de la misma forma puede evitar la depresión y la ansiedad.
- Ayuda para combatir la Esquizofrenia: Según un estudio realizado por el Centro de Neurociencia Cognitiva de Lyon (Francia), los pacientes esquizofrénicos que jugaban a juegos de estrategia de forma diaria mostraban un estado de mejoría mostrando mejor atención, planificación y razonamiento.
- Mejora el razonamiento y la resolución de problemas: Normalmente, le gente que juega a algún juego de estrategia aprende a jugar de pequeño, y realmente es el mejor momento, ya que según investigaciones al respecto, jugar a juegos de estrategia mejora el pensamiento y la resolución de problemas de un niño, la lectura y los resultados matemáticos.
- Aumenta la autoestima: En los juegos de estrategia sueles ir por tu cuenta y utilizar mucho la mente. Si pierdes, puedes buscar tus errores y analizar donde has fallado, porque no has ganado, y poner más voluntad y fuerza mental la próxima vez.
- Ayuda en terapia y rehabilitación: De la misma forma que algunos estudios demuestran que los videojuegos pueden ser muy útiles para la terapia con pacientes que han sufrido un ataque cerebrovascular, los juegos de estrategia también pueden servir para ello. Incluso se pueden usar en terapia para individuos con autismo o discapacidades varias. Mover las piezas por el tablero puede ayudar como terapia motora, mientras que

contribuye a realizar un esfuerzo mental extra y provoca un aumento de la concentración, incluso llegando a relajar a los pacientes.

- Eleva el coeficiente intelectual: Diversos son los estudios que han llevado a la conclusión de que los juegos de estrategia mejoran el coeficiente intelectual. Por nombrar un par de ellos, existe un estudio llevado acabo con 4000 estudiantes venezolanos donde se comprobó que después de 4 meses jugando casi diariamente a juegos de estrategia su coeficiente intelectual había aumentado. En otro estudio llevado a cabo por el doctor Peter Dauvergne de la universidad de Sidney, este concluye que los juegos de estrategia mejoran la capacidad de resolver problemas, mejoran las habilidades lectoras, el lenguaje, las matemáticas y la memoria, ayudan a desarrollar un pensamiento creativo y original, ayudan a aprender a tomar decisiones más precisas y rápidas bajo presión, ayudan a mejorar las notas en los exámenes, ayudan a aprender a elegir mejor entre varias opciones, ayudan a la concentración y un largo etcétera, y todo ello independientemente del sexo o nivel socioeconómico de los jugadores. Llegando a la conclusión final de que mejoran el coeficiente intelectual en general.
- Mejora la creatividad: El hemisferio derecho del cerebro es el responsable de la creatividad, teniendo en cuenta lo dicho anteriormente en el beneficio de que los juegos de estrategia ejercitan ambos hemisferios cerebrales, no debe sorprender que los juegos de estrategia ayuden a desarrollar la creatividad. De hecho existen nuevamente distintos estudios que así lo demuestran. De entro todos ellos, cabe destacar el estudio del doctor Ferguson, realizado con estudiantes, en el cual transcurridas 32 semanas, el grupo de alumnos obtuvo mejores resultados en las pruebas de creatividad, con la originalidad como principal mejora de sus aptitudes.
- Potencia la memoria: Este beneficio, después de haber leído los anteriores, de nuevo es una obviedad, pero a pesar de ello haremos referencia a un viejo estudio realizado en 1985 donde se demostraba que los estudiantes que practicaban juegos de estrategia destacaban por su mejor memoria en todas las asignaturas. Otro experimento realizado más recientemente en Pensilvania comprobó que los alumnos que nunca habían jugado a juegos de estrategia también mejoraban notablemente su memoria y sus habilidades verbales (lo que tiene gracia teniendo en cuenta lo poco que se suele hablar durante una partida) tras empezar a jugar.
- Incrementa la capacidad lectora: Este beneficio podría considerarse de sorprendente, pero un estudio realizado por el doctor Stuart Margulies en 1991 en 53 colegios de educación primaria de Nueva York, demostró que los chavales que participaron en el programa de ajedrez (juego de estrategia), durante dos años, mejoraron de forma significativa su capacidad lectora y superaron la media nacional. La ventaja media de los jugadores fue de 5.4 puntos en el percentil nacional. El propio Margulies expone algunas teorías

para justificar esta propiedad 'milagrosa', pero no ofrece una conclusión definitiva.

- Facilita la concentración: Un beneficio que sin duda no sorprende es este, puesto que los juegos de estrategia exigen un alto grado de concentración.
- Enseña a planificar y hacer previsiones: La corteza prefrontal es una de las ultimas zonas del cerebro en desarrollarse (los adolescentes todavía son inmaduros en este campo), justo el área responsable de planificar y anticiparse a los acontecimientos, del autocontrol y el buen juicio. Pues bien, los juegos de estrategia se han revelado como una forma magnifica de desarrollar la corteza prefrontal y ayudar a tomar mejores decisiones en cualquier área de la vida.

Por tanto, tal y como se anunció al principio de esta subsección, los juegos de estrategia no son solo buenos para los niños sino también para las personas adultas. De hecho un estudio reciente ha relacionado el juego de estrategia con la mejora de "procesos ejecutivos de control" (mejora de las capacidades mentales) en las personas mayores.

3.2.2. El juego de las Damas:

Las damas es un juego de mesa de estrategia para dos contrincantes que se juega por turnos.

Se usa un tablero con cuadros de dos colores alternativamente.

Las fichas son pequeñas piezas redondas. Cada contrincante tiene las fichas de un color diferente. Estas fichas se pueden mover verticalmente tanto hacia la derecha como hacia la izquierda.

Pero existen varias modalidades del juego de las damas que requieren distintos tableros, distinto número de piezas o incluso distinto color de las piezas. Por mencionar algunas modalidades, podemos encontrar las damas españolas, las damas rusas, las damas turcas, las damas chinas, las poddavki, etc...

Para este proyecto elegiremos la modalidad de las damas españolas, por tratarse de una de las variantes de las damas más jugadas mundialmente.

3.2.2.1. Las Damas españolas

Las damas españolas son, no solamente una variante del juego de las damas, sino que es la variante de damas más jugada en el mundo.

Se juega sobre un tablero de 8x8 casillas con 12 fichas por cada jugador, siendo las fichas de un jugador de color blanco y las del otro jugador de color negro.

Existen varias reglas específicas respecto al movimiento y captura de las fichas:

- Las Damas o peones se mueven diagonalmente una casilla hacia delante.
- Las Damas o peones capturan también sólo diagonalmente hacia delante.
- Las Reinas o peones promocionados mueven diagonalmente hacia delante o hacia atrás cualquier número de casillas.
- Las Reinas o peones promocionados capturan también diagonalmente hacia delante o hacia atrás cualquier número de casillas.
- Las piezas sobre las que saltan, son retiradas del tablero después de efectuar el movimiento.
- Cuando una dama o peón llega hasta la última fila del lado contrario, se convierte en reina o peón promocionado.
- Para poder capturar, la casilla contigua a la casilla donde se encuentra la ficha que se va a capturar debe de estar libre (sin ninguna ficha, ni propia, ni del adversario).
- No se puede saltar una ficha sin capturar, por tanto no se pueden saltar tus propias fichas.

Una partida finaliza cuando:

- Un jugador se queda sin piezas sobre el tablero, por tanto ha perdido.
- Un jugador no puede mover llegado su turno, puesto que todas las fichas que le quedan sobre el tablero están bloqueadas, por tanto ha perdido.
- Ambos jugadores quedan con todas las fichas bloqueadas, por tanto quedan en tablas.

Tablero de damas españolas

3.3. Inteligencia Artificial (IA) y los juegos de mesa

La inteligencia artificial o IA por sus siglas, es un área de la informática que profundiza en la creación de máquinas inteligentes capaces de trabajar y reaccionar como humanos (reconociendo el habla, percibiendo el ambiente, manipulando objetos, razonando, aprendiendo, planeando y resolviendo problemas, entre otras características).

Sólo con saber que es la inteligencia artificial, se te vienen millones de posibles aplicaciones de la IA, o campos donde pueda estar actualmente trabajando. Pero seamos más concretos, ¿Qué aplicaciones tiene las Inteligencia artificial?

3.3.1. Aplicaciones de la IA

Actualmente hay infinidad de aplicaciones donde no sólo está presente, sino que es fundamental la inteligencia artificial, algunas de ellas se citan a continuación:

- Procesamiento de lenguajes naturales: Capacidad de traducción, órdenes a un sistema Operativo, conversación hombre-máquina.
- Sistemas expertos o de apoyo a la decisión: Sistemas que se les implementa experiencia para conseguir deducciones cercanas a la realidad.
- **Robótica:** Navegación de robots móviles, control de brazos móviles, ensamblaje de piezas, etc… La IA tiene aplicación en la robótica cuando se requiere que un robot “piense” y tome una decisión entre dos o más opciones, es entonces cuando principalmente ambas ciencias comparten algo en común.
- Problemas de percepción: Visión y habla, reconocimiento de voz, obtención de fallos por medio de la visión, diagnósticos médicos, etc…
- Aprendizaje: Modelización de conductas para su implante en computadoras.
- Medicina: Para diagnósticos, tratamientos, etc…
- Minería de datos: búsqueda de patrones en grandes volúmenes de conjuntos de datos.
- Mecatrónica: es un mix de ingeniería mecánica, ingeniería electrónica, ingeniería de control e ingeniería informática.
- Juegos: Dotando de cierta inteligencia a los agentes adversarios.
- Mundos virtuales.
- Etc…

Así pues, y haciendo referencia a conceptos explicados anteriormente, se deduce, que la robótica es, ni más ni menos, que una de las posibles aplicaciones de la inteligencia artificial.

3.3.2. ¿Por qué es importante la robótica y la IA?

Como hemos podido ver, la robótica es un campo emergente con aplicaciones en muchas facetas de nuestra vida diaria. Es importante para todos los miembros de la sociedad entender la tecnología y todo lo que ello conlleva. Pero, la robótica y la inteligencia artificial son importantes por mucho más que ello, ya que proporcionan una combinación única de ciencia, tecnología, ingeniería y matemáticas.

3.3.3. Algoritmos para juegos

Una vez ya hemos relacionado los juegos con sus respectivos beneficios con la robótica y la inteligencia artificial, se hace necesario profundizar en cómo aplicar la inteligencia artificial a los juegos, y más concretamente a los juegos de mesa. En este punto, se puede descubrir que la inteligencia artificial actualmente esta muy presente en el desarrollo de juegos.

Los juegos con inteligencia artificial se pueden clasificar en juegos de tiempo real y juegos por turnos, estos últimos que son los que nos interesan para este proyecto, a su vez se pueden sub-clasificar en función del número de jugadores como juegos de múltiples jugadores (como por ejemplo el parchís) o juegos de únicamente dos jugadores (como es el caso de las damas).

Los juegos son una simplificación de la realidad, en la cual todas las cualidades de la supuesta realidad se convierten en un conjunto finito de reglas, perfectamente definidas, y sin ninguna clase de incertidumbre (quizás convenga aclarar que un elemento aleatorio de probabilidad conocida no es incierto, sino aleatorio, como por ejemplo lanzar un dado).

Por tanto la IA puede abordar juegos de manera relativamente sencilla, y con gran éxito.

Un juego es una situación conflictiva en la que uno debe tomar una decisión sabiendo que los demás también toman decisiones, y que el resultado del conflicto se determina, de algún modo, a partir de todas las decisiones realizadas. (“John Von Neumann”).

Para llevar a cabo el comportamiento “inteligente” se han desarrollado gran variedad de métodos o mecanismos. En la mayoría de los casos, dado una entrada (situación de la realidad, equivalente a lo que el humano recibiría por sus sentidos), el sistema debe devolver una salida (acción que desempeña el humano).

En concreto, se suele buscar una solución en un conjunto de “estados solución” posibles. Para encontrar dicha solución hay infinidad de métodos entre los que están:

- Búsqueda en un espacio de estados clásica: Búsqueda en profundidad, búsqueda en anchura, búsqueda voraz (Greedy), algoritmo A*, etc…

- El uso del paradigma basado en reglas: Se parte de un estado inicial, al cual se le aplican reglas hasta obtener el estado de salida (solución).
- Paradigma de razonamiento basado en casos: Se parte de un conjunto de problemas ya resueltos. Dado un nuevo problema, se busca el problema más similar del conjunto de problemas resueltos (etapa de recuperación), y se devuelve la adaptación de la solución del problema más similar (adaptación). Si el nuevo caso o problema resuelto es correcto y representativo (etapa de revisión), se aprende (etapa de aprendizaje).
- Programación evolutiva: Dado un conjunto de soluciones escogidas inicialmente de forma aleatoria, estas se irán cruzando, mutando y seleccionando a lo largo de distintas generaciones, imitando un proceso natural de selección de Darwin.
- Búsqueda en situaciones con dos adversarios: Minimax, alfa-beta. Son los más usados en juegos de adversarios como por ejemplo el ajedrez.
- Y muchos más…

La mayoría de los juegos que se estudian en IA son deterministas porque la suerte no interviene, se juegan alternativamente por turnos, generalmente de dos jugadores, con información perfecta, es decir, los juegos son conocidos y limitados, cada jugador conoce perfectamente la evolución del juegos y los movimientos que puede hacer su oponente (porque son observables), y además la ganancia (o perdida) de un participante es exactamente la perdida (o ganancia) del oponente, de manera que esta oposición define los juegos como problemas de búsqueda con adversario.

Como para nuestro proyecto necesitamos implementar un algoritmo para el juego de las damas (juego con dos adversarios), nos centraremos en estos últimos métodos, es decir, algoritmo Minimax y algoritmo de poda Alfa-Beta.

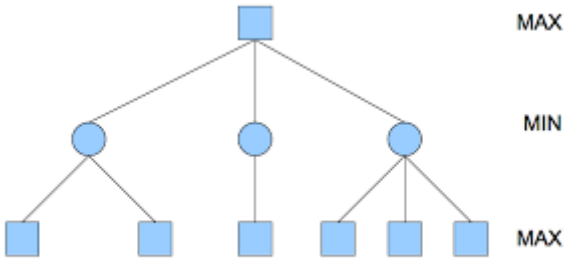
Siempre existe una forma racional de actuar en juegos de dos participantes, si los intereses que los gobiernan son completamente opuestos. (“John Von Neumann”).

3.3.3.1. Minimax

El algoritmo Minimax se utiliza solamente en juegos en los que no influya el azar (como en juegos de cartas), sino en lo que denominamos juegos perfectamente informados (es decir, que tenemos toda la información tanto de nuestra jugada como la del adversario).

El algoritmo Minimax construye un árbol partiendo del estado actual del juego, un nodo hijo por cada posible movimiento legal que puede realizar el ordenador (se llama al ordenador jugador MAX). En el siguiente nivel, por cada posible movimiento del ordenador, se genera los posibles movimientos legales que podría realizar el jugador humano (se llama jugador MIN al usuario jugador), y así sucesivamente hasta llegar a un nodo que llamaremos terminal, en el que haya un desenlace valido, es decir, que en cada nivel del árbol se van alternando el jugador MAX y el jugador MIN (de ahí, el nombre de estrategia Minimax).

Si representamos con cuadrados los nodos MAX y con círculos los nodos MIN, un posible despliegue de un árbol Minimax sería el siguiente:

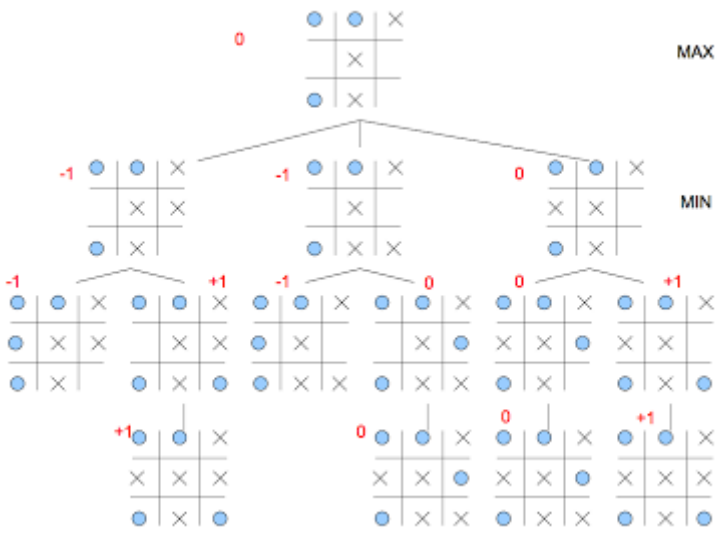


Árbol de búsqueda minimax

Cada vez que se llega a un nodo terminal, se usa lo que se llama función de evaluación o utilidad, que no es más que una función que sopesa que tan buena es la solución que conduce a ese nodo terminal, asignándole un valor numérico.

Lo mejor es ilustrarlo con un ejemplo. Debido a la simplicidad, lo ejemplificaremos a través del juego tic-tac-toe (o tres en raya).

Para este juego, una posible función de utilidad seria por ejemplo sumar 1 punto a los nodos terminales en lo que gana el ordenador, sumar cero putos a los nodos en los que se produce un empate y restar un punto a los nodos en los que gana el jugador humano. En el siguiente esquema podemos ver un posible despliegue Minimax para el juego del tic-tac-toe:



Árbol de búsqueda Minimax en el juego de tic-tac-toe

El algoritmo evalúa cada nivel del árbol de abajo hacia arriba, y en los niveles MAX elegirá los caminos que conduzcan a puntuaciones más altas, que son las que favorecen al ordenador, y en los niveles MIN tratara de escoger aquellos caminos con menor puntuación, ya que penalizan al jugador humano. La función de evaluación recorre el árbol hasta llegar a los nodos terminales, y los puntúa tal y como habíamos acordado anteriormente.

En la imagen se puede observar que en el último nivel, se marca con +1 punto el nodo inferior derecho y el nodo inferior izquierdo (gana MAX) y con +0 los nodos centrales (los jugadores quedan en tablas). Los nodos inmediatamente superiores son nodos MAX, por lo que puntuaremos los estados con el mayor valor de los hijos (en este caso, los que tienen descendencia solo tienen un hijo). Vemos que en este nivel hay dos nodos terminales en los que gana MIN, así que los marcamos con -1 punto. El siguiente nivel inmediatamente superior es nivel MIN, por lo que seleccionaremos las menores puntuaciones de los nodos hijos. En el estado de la izquierda nos quedaremos con -1 que es el menos, en el central también nos quedaremos con -1 porque es el menos, y en el de la derecha el menos valor es +0. Finalmente, en el nodo raíz, es un nodo MAX, por lo que seleccionara el mayor de los valores de los nodos hijos, en este caso el mayor valor es el valor +0. Es decir, el siguiente movimiento del ordenador será el que esta etiquetado con 0, por lo que pondrá la X en la casilla lateral izquierda de la línea central.

A continuación seria el turno del jugador humano, y una vez el jugador humano haya realizado su movimiento, con el nuevo estado del tablero, el ordenador volverá a realizar la misma operación para volver a elegir el siguiente movimiento, y así sucesivamente hasta alcanzar un estado final (ganar, perder o empatar).

El algoritmo en pseudocódigo quedaría más o menos así:

```
Minimax(jugador, tablero)

Si es nodo terminal, devolver ganador

Sino

    Nodos hijos = todos los movimientos legales desde
    el estado actual

    Si es turno de MAX:

        Devolver valor máximo de llamada a Minimax()
        para cada nodo hijo

    Sino

        Devolver valor mínimo de la llamada a
        Minimax() para cada nodo hijo
```

Como se puede observar se trata de una implementación recursiva de un recorrido en profundidad.

Por tanto, el algoritmo Minimax genera un árbol del juego en anchura hasta un cierto nivel de profundidad, en el nivel máximo de profundidad elegido se generan los nodos terminales, a continuación se aplica la función de utilidad a cada nodo terminal y se usan los valores de utilidad de los nodos terminales para determinar la utilidad de los nodos predecesores. Se vuelcan los valores de utilidad desde los nodos hoja hasta el nodo raíz, nivel por nivel. Volcando el menos valor de utilidad en los nodos MIN, y el mayor valor de utilidad en los nodos MAX.

Pero, incluso para juegos sencillos, realizar una búsqueda completa es inviable (el tamaño de árbol de búsqueda es muy grande).

Para ello se proponen dos soluciones:

1. Incrementar la información heurística del método aun a costa de perder admisibilidad. El objetivo es aproximar b* = 1. (Esto no es viable en algunos juegos ya que se dispone de poca información sobre el problema).

2. Podar el árbol de búsqueda en ciertos puntos y usar métodos para seleccionar el mejor movimiento.

Y de esta última opción, se genera el algoritmo Alfa-Beta.

3.3.3.2. Poda Alfa-Beta

Poda Alfa-beta es una técnica de búsqueda que reduce el número de nodos evaluados en un árbol de juego por el algoritmo Minimax.

Como acabamos de comentar, el problema del algoritmo de búsqueda Minimax es que el número de estados a explorar es exponencial al número de movimientos. Partiendo de este hecho, la técnica de poda Alfa-Beta trata de eliminar partes grandes del árbol, aplicándolo a un árbol Minimax estándar, de forma que se devuelva el mismo movimiento que devolvería este, gracias a que la poda de dichas ramas no influye en la decisión final.

La poda alfa-beta toma su nombre de la utilización de dos parámetros que describen los limites sobre los valores hacia atrás que aparecen a lo largo de cada camino.

- α es el valor de la mejor opción hasta el momento a lo largo del camino para MAX, esto implicara por lo tanto la elección del valor más alto.
- β es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, esto implicara por lo tanto la elección del valor más bajo.

Esta búsqueda alfa-beta va actualizando el valor de los parámetros según se recorre el árbol. El método realizara la poda de las ramas restantes cuando el valor actual que se está examinando sea peor que el valor actual de α o β para MAX o MIN, respectivamente.

El desarrollo del algoritmo en pseudocódigo quedaría más o menos así:

```
  Función alfa-beta(nodo, profundidad, α, β,
jugadorMaximizador):
        SI nodo es un nodo terminal o profundidad = 0
              Devolver el valor heurístico del nodo
        Si jugadorMaximizador
              Para cada hijo de nodo:
                    α := max(α, alfa-beta(hijo, profundidad
- 1, α, β, FALSE))
```

```
                    Si β <= α

                        Break (* poda β *)

            Devolver α

        SiNo

            Para cada hijo de nodo:

                β := min(β, alfa-beta(hijo, profundidad
- 1, α , β, TRUE))

                    Si β <= α

                        Break (* poda α *)

            Devolver β
```

Siendo la llamada inicial:

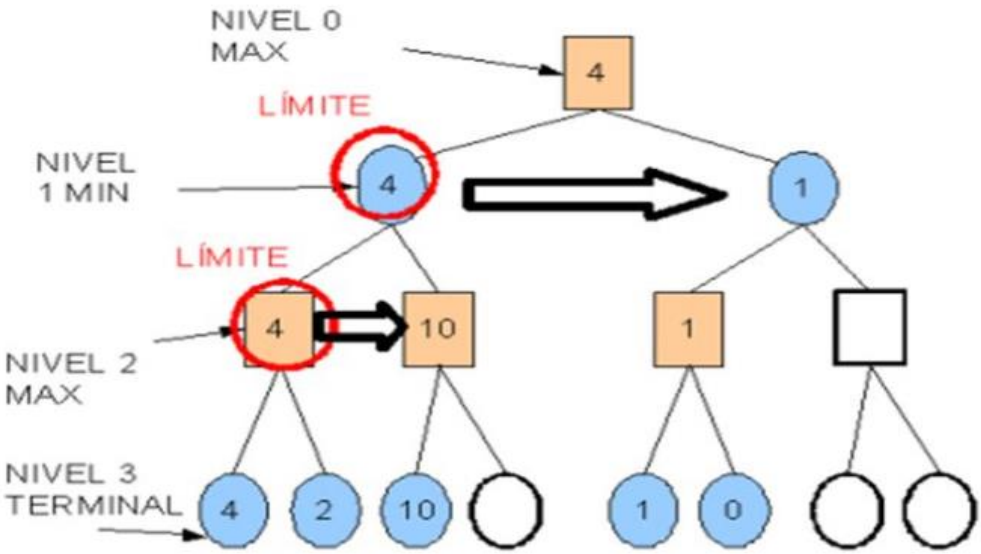
```
        Alfa-beta(origen, profundidad, -infinito,
+infinito, TRUE)
```

La eficacia de la poda alfa-beta depende del orden en el que se examinan los sucesores, es decir, el algoritmo se comportara de forma más eficiente si examinamos primero los sucesores que probablemente serán los mejores.

Si esto pudiera hacerse, implicaría que alfa-beta solo tendría que examinar $O(b^{d/2})$ en lugar de los $O(b^d)$ de Minimax. Esto implica que el factor de ramificación eficaz será de $\sqrt{b}$ en lugar de b. En otras palabras, alfa-beta podría mirar hacia delante aproximadamente dos veces más que Minimax en la misma cantidad de tiempo.

Si se recurre a una ordenación aleatoria en lugar de primero el mejor de los sucesores, el número aproximado de nodos examinados seria de $O(b^{3d/4})$ para un valor moderado de b. En el ajedrez por ejemplo se puede realizar una función de ordenación sencilla teniendo en cuenta primero la captura de fichas, después amenazas, movimientos hacia delante y por último movimientos hacia detrás, esto conseguiría aproximadamente un factor de dos del resultado $O(b^{d/2})$ del mejor caso. La inclusión de esquemas dinámicos para ordenar movimientos, basados en experiencia podrían acercarse al límite teórico.

De nuevo, lo mejor es ilustrarlo con un ejemplo.



Árbol de búsqueda Minimax con poda Alfa-Beta

3.4. 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.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

4. Trabajo realizado

Tras una búsqueda exhaustiva de información, en esta sección se analizará y comentará el desarrollo del proyecto, desde la toma de decisiones hasta el diseño e implementación del algoritmo e integración en el robot NAO.

4.1. Planteamiento del proyecto

Teniendo en cuenta las motivaciones que nos llevan a desarrollar este proyecto, nos encontramos con la necesidad de implementar el juego de las damas en un robot bípedo NAO, y para ello debemos hacer uso de los avances y conocimientos que disponemos en inteligencia artificial aplicada al campo de la robótica. Para conseguirlo, utilizaremos robots humanoides NAO que nos permitirán romper algunas de las barreras que la gente aún tiene para sentarse a jugar hacia la pantalla del ordenador, logrando además un clima más similar a lo que podría ser una partida en el salón de casa con algún familiar o amigo.

4.1.1. Toma de decisiones

En base a la información recolectada en el apartado anterior 'Trabajo relacionado', se llega a la decisión de utilizar la modalidad de las damas españolas, el lenguaje de programación Python, y crear un algoritmo de búsqueda Minimax para la implementación del algoritmo que permitirá al robot NAO jugar a las damas.

4.1.2. Las damas españolas

La decisión de utilizar la variante de las damas españolas ha sido tomada en base a dos razones principales. La primera de ellas, es que las damas españolas, es la variedad de las damas más extendida mundialmente, y por tanto la variedad que más gente sabe jugar. La segunda razón, es que contiene un conjunto de reglas bastante variadas en cuanto a movimientos, y tanto el tamaño del tablero como la cantidad de fichas en cada partida es comúnmente la más utilizada en la mayoría de las variedades del juego de las damas.

4.1.3. Programación con Python

Para el desarrollo de este proyecto se utilizará el lenguaje de programación Python porque es el lenguaje utilizado originalmente en las herramientas y aplicaciones proporcionadas por la empresa Aldebaran, como por el ejemplo Choregraphe. Si bien es cierto que existe la posibilidad de utilizar otros lenguajes de programación, en el momento de tener que editar algún script proporcionado por la empresa, siempre será

más fácil utilizando el lenguaje de programación Python, o como mucho C++ que cualquier otro de los lenguajes con los que tenemos la posibilidad de trabajar al programar los robots NAO.

Como valor añadido a la elección tomada, destacar que además, Python es el lenguaje más utilizado por otros desarrolladores de robots NAO, lo que facilitará la incorporación de nuevas funcionalidades futuras desarrolladas incluso por otros desarrolladores si es que se desease poder utilizar esta opción.

4.2. Desarrollo del programa principal para jugar a las damas

Puesto que el proyecto consiste en la implementación del juego de las damas en un robot NAO, podemos definir el desarrollo del algoritmo del juego de las damas, como el "core" del proyecto.

Lo primero será elaborar un esquema sobre las necesidades, funcionalidades, etc... Que a groso modo, el algoritmo va a necesitar, para luego entrar más detalladamente en cada una de esas funciones.

Al tratarse de un juego con adversario por turnos sabemos que nuestro programa deberá constar de dos acciones básicas, realizar jugada y esperar la jugada del adversario.

Un detalle a tener en cuenta, es que si un jugador A, captura la ficha del otro jugador B, el siguiente movimiento no corresponderá al jugador B, sino al jugador A.

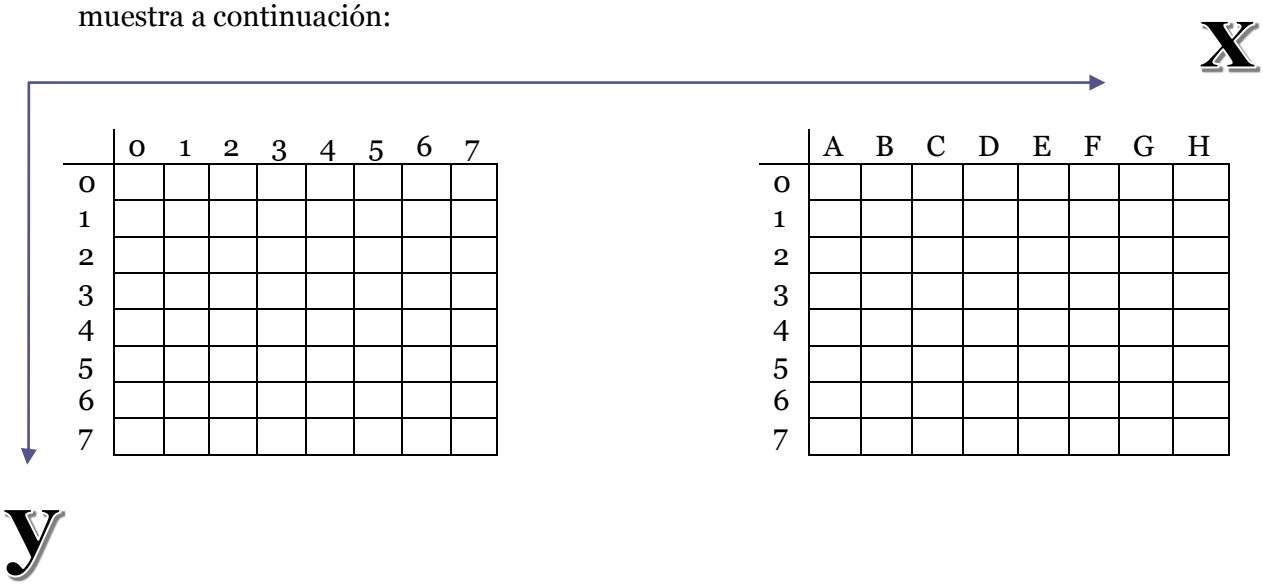
Por tanto inicialmente, se debe crear un bucle inicial que con pseudocódigo quedaría más o menos así:

```
While (no exista ganador, perdedor o empate):
      Realizar jugada
      if (no se ha capturado ficha):
            Do:
                  Esperar jugada del oponente
            While (el jugador B (o oponente) capture ficha)
```

Una vez perfectamente estructurado el concepto básico de lo que sería una partida de damas, es momento de entrar más en detalle en las 2 funciones principales de las que consta el programa, realizar jugada y esperar jugada del oponente.

Empecemos por la función más fácil, la función esperar jugada del oponente. Esta será una función bastante básica en la que el robot NAO quedará a la espera de saber el movimiento realizado por su oponente, tal movimiento será transmitido al robot NAO a través de la voz, mediante un conjunto de coordenadas que previamente habremos

definido, del estilo (x,y), siendo X la coordenada sobre el eje horizontal, e Y la ordenada sobre el eje vertical, en un espacio bidimensional cartesiano; para ello podremos utilizar letras o números, quedando un tablero con coordenadas del modo que se muestra a continuación:



Para que la partida pueda llegar a buen puerto, el robot NAO no solo necesitará saber a qué posición va a mover su oponente, sino que también deberá tener constancia de qué ficha es la que se está moviendo, por ello, finalmente, el adversario deberá comunicar verbalmente tanto las coordenadas de origen (donde se encuentra la ficha que va a mover) como las coordenadas destino (hasta donde se va a mover la ficha).

Por otra parte, tenemos pendiente la función Realizar jugada. Está función ya será bastante más compleja, pues es en este momento cuando el robot debe analizar la situación actual del tablero, tomar una decisión sobre que ficha mover, y mover la ficha haciéndoselo constar al adversario a través de la voz nuevamente.

Por tanto un posible pseudocódigo sería:

```
Función realizarJugada:

     Recoger información actual del tablero (o situación del
juego)

     Algoritmo Minimax

     Mover ficha

     Dar constancia al adversario de la ficha movida
```

La información del tablero, o de la partida en general está almacenada en la memoria del programa ejecutado por el robot, por tanto en cada llamada a la función realizarJugada se deberá llamar al algoritmo Minimax desarrollado, proporcionándole dicha información. Tras la ejecución del algoritmo minimax, éste deberá devolver un valor que permita conocer cuál es la mejor jugada posible dado el estado actual de la partida, este valor obtenido (mejor movimiento posible), será transmitido por el robot al adversario a través de la voz, como se dijo anteriormente para ello se deberá de

expresar tanto las coordenadas de origen como las coordenadas destino. Además, se utilizará dicho valor, el mejor movimiento posible, para llamar a la función moverFicha que será la encargada de modificar la situación actual de la partida modificando los estados que generen ese nuevo movimiento.

4.3. Algoritmo de búsqueda con adversario, Minimax

El algoritmo Minimax, como hemos dicho anteriormente, se encargará de calcular la jugada óptima a partir de una situación determinada de la partida.

El algoritmo Minimax consiste pues, en generar un árbol de todas las jugadas posibles a las cuales se les asigna una puntuación, el objetivo consiste en seleccionar la jugada con la mejor puntuación.

El algoritmo tiene un parámetro que establece la profundidad del árbol y es el factor que nos permite establecer un compromiso entre el tiempo de búsqueda y la calidad de las jugadas.

Para poder seleccionar fácilmente el nivel de profundidad del algoritmo de búsqueda, se ha creado una variable contador que va incrementándose en cada nivel de profundidad del algoritmo.

4.3.1. Max - Nivel 1:

Teniendo en cuenta el estado actual de la partida (o el supuesto estado de la partida, si se accede a la función nivel 1 desde la función Min, nivel 2), el robot NAO intentará con cada una de las fichas que no tiene bloqueadas, todos los movimientos posibles. Si alguno de estos movimientos capturase una ficha del adversario, para ese movimiento en concreto, y con el nuevo supuesto estado de la partida, el robot NAO volverá a intentar mover todas las fichas que no tiene bloqueadas en todas las direcciones posibles. Una vez desplegados todos los nodos correspondientes a todos los posibles movimientos a los que se puede llegar dado el estado actual del juego, se comprobará la variable contador, de manera que la función Max podrá actuar de dos formas distintas en función del resultado de la comparación:

1. Si la variable contador ha llegado a su máximo valor: para cada uno de esos posibles nuevos estados obtenidos en cada movimiento, se calculará una función de utilidad (que explicaremos más adelante);
   Con los resultados obtenidos de la función de utilidad de todos los nuevos posibles estados de la partida, se hará una comparación para finalmente aceptar que el mejor movimiento es aquel con el valor de la utilidad más alto.

2. Si la variable contador aún no ha alcanzado su valor máximo, entonces, para cada uno de esos nuevos estados generados por cada movimiento posible se hará una llamada al nivel 2 (o nivel Min) del algoritmo Minimax.

Recogiendo así los valores devueltos por el nivel 2, para posteriormente compararlos, y nuevamente quedarse con el movimiento con el valor de mayor de utilidad.

En ambos casos, tras decidir cuál es el mejor movimiento, se almacenarán los valores del mejor movimiento equivalentes a la dirección del movimiento, las coordenadas (x,y) iniciales, el índice de posiciones que se moverá la ficha y el valor de la utilidad. A continuación se volverá a hacer una comprobación, en este caso se comprobará si la profundidad es igual a uno, en cuyo caso, al encontrarse al principio del algoritmo, la función devolverá un array que contiene todos los valores almacenados (coordenadas, índice de posiciones a mover, la dirección y el valor de la utilidad) referentes al mejor movimiento. En caso contrario, tan solo devolverá el valor de la utilidad del mejor movimiento.

4.3.2. Min - Nivel 2:

Al nivel Min siempre se accede desde el nivel Max (nivel 1). Pasándole como parámetros los supuestos estados en los que se encuentra la partida, a partir de cada uno de estos supuestos estados en los que se encontraría la partida tras mover su ficha el robot, el robot NAO intentará simular todos los posibles movimientos que podría realizar su oponente con todas las fichas que no tenga bloqueadas. Si en alguno de estos posibles movimientos, la ficha del oponente capturase alguna ficha del robot, para ese movimiento en concreto y con el nuevo supuesto estado de la partida, el robot NAO volverá a intentar simular todos los posibles movimientos que podría realizar el oponente con las fichas que no tuviese bloqueadas. Una vez desplegados los nodos correspondientes a todos los posibles movimientos del oponente, nuevamente se comprobara el valor de la variable contador, de manera que la función Min también podrá actuar de dos formas distintas en función de la comparación de esta variable contador:

1. Si la variable contador ha llegado a su máximo valor: Para cada uno de esos nuevos posibles estados de la partida obtenidos, se calculará la función de utilidad.
   Con los resultados de la función de utilidad obtenidos en cada posible estado de la partida, se hará una comparación, para esta vez, aceptar que el mejor movimiento será aquel que en donde la utilidad obtenida tenga un valor menor.
2. Si la variable contador aún no ha alcanzado su máximo valor: Para cada uno de esos nuevos posibles estados generados se hará una llamada a la función del nivel 1 (o nivel Max) del algoritmo Minimax.
   Para así posteriormente recoger los valores devueltos por el nivel 1, compararlos y quedarse con aquel en el que la utilidad tenga un valor menor.

En ambos casos, tras decidir cuál es el mejor movimiento, es decir, el movimiento con el valor de utilidad más bajo, se almacenará dicho valor. Y este valor será exactamente el valor que devolverá la función del nivel 2 (o nivel Min).

Con esto creamos un bucle, tal que, mientras no se llegue al valor de profundidad deseado, la función del nivel Max llamará a la función del nivel Min, y la función del

nivel Min llamará a la función del nivel Max, y así sucesivamente eligiendo el valor mínimo de utilidad en la función Min y el valor máximo de utilidad en la función Max, para finalmente obtener los datos necesarios para poder realizar el mejor movimiento posible de todos los movimientos disponibles en el momento dado de la partida en el que se ejecuten las funciones Min y Max.

Para poder entenderlo mejor, a continuación se mostrará el pseudocódigo de ambas funciones con un poco más de detalle:

```
funcionMax (estado de la partida):

  For (cada ficha del robot en el tablero):

      If (la ficha no está bloqueada O si al moverla no se
sale del tablero):

          probar todos los posibles movimientos de la
ficha

          If (el movimiento captura ficha):

              utilidad = funcionMax(supuesto estado
actual de la partida)

          else:

              calcular la utilidad de todos los
movimientos posibles con esa ficha

          If (profundidad == profundidad total):

              comparar los valores de utilidad y
almacenar, la dirección, las coordenadas de la ficha y el valor
de la utilidad, del mayor valor de utilidad obtenido

          else:

              For (cada nuevo estado obtenido de los
distintos posibles movimientos de la ficha):

                  utilidad = funcionMin(supuesto estado
actual de la partida)

                  comparar los valores de utilidad y
almacenar la dirección, las coordenadas de la ficha y el valor
de la utilidad del mayor valor de utilidad obtenido

  If (profundidad == 1):

      Return [dirección, coordenadaX, coordenadaY,
utilidad]
```

```
    else:
        Return utilidadMaxima
```

funcionMin(estado de la partida):

```
For (cada ficha del adversario en el tablero):
    If (la ficha no está bloqueada O si al moverla no se sale del tablero):
        probar todos los posibles movimientos de la ficha
        If (el movimiento captura ficha):
            utilidad = funcionMin(supuesto estado actual de la partida)
        else:
            calcular la utilidad de todos los movimientos posibles con esa ficha
        If (profundidad == profundidad total):
            comparar los valores de utilidad y almacenar el menor valor de utilidad obtenido
        else:
            For (cada nuevo estado obtenido de los distintos posibles movimientos de la ficha):
                utilidad = funcionMax(supuesto estado actual de la partida)
                comparar los valores de utilidad y almacenar el menor valor de utilidad obtenido
Return utilidadMinima
```

4.3.3. Calculo de la función de utilidad:

La función de utilidad utilizada en este algoritmo se basa en los siguientes cálculos:

Utilidad = [((Número de damas del robot en el tablero) * 100) +

((Número de damas del adversario en el tablero) * (-100)) +

((Número de reinas del robot en el tablero) * 250) +

((Número de reinas del adversario en el tablero) * (-250)) +

((Cada ficha del robot en el tablero que este bloqueada) * (-5)) +

((Cada ficha del adversario en el tablero que este bloqueada) * 5) +

((Cada ficha del robot en el tablero que NO este bloqueada) * 2) +

((Cada ficha del adversario en el tablero que NO este bloqueada) * (-2))]

4.3.4. Poda Alfa-Beta:

Se ha implementado una poda Alfa-Beta al algoritmo Minimax con la finalidad de reducir el espacio de búsqueda eliminando aquellas ramas del árbol que no van a producir mejores resultados, para así lograr reducir el tiempo de cómputo.

La poda Alfa-Beta consiste en podar el árbol de búsqueda desarrollado por el algoritmo, en esta ocasión y a modo de prueba, el código añadido al algoritmo Minimax consiste en analizar a lo largo de las distintas iteraciones o niveles, los distintos movimientos posibles, comprobando la utilidad resultante de realizar los movimientos, para poder compararlas y finalmente pasar al siguiente nivel, sólo con aquellos movimientos que hallan devuelto los mejores valores de utilidad, concretamente los tres mejores.

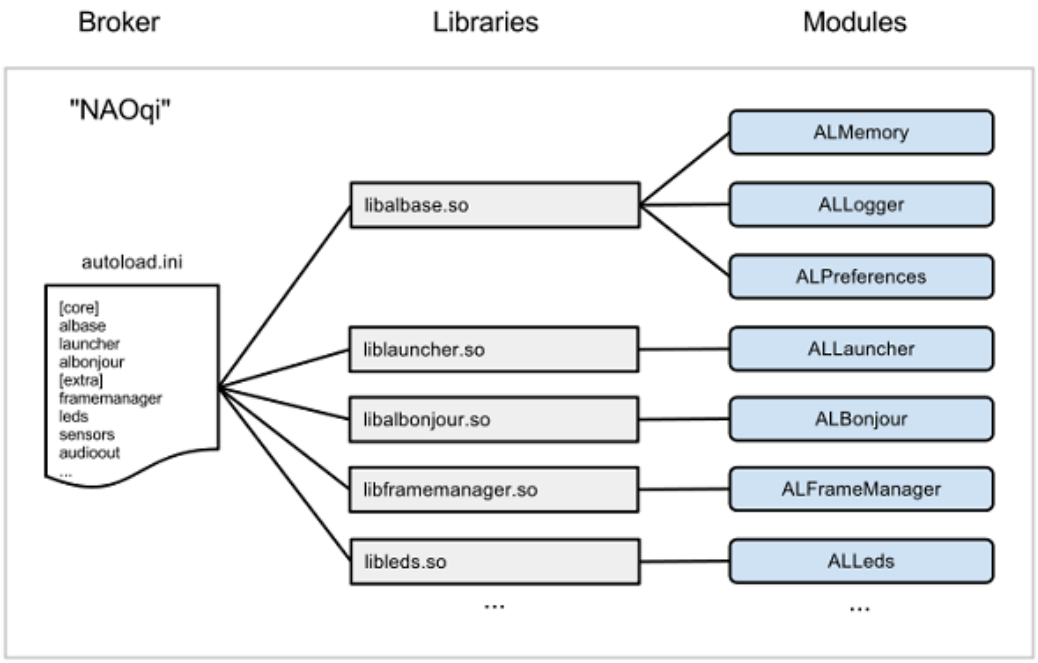
Es decir, tras analizar todos los posibles movimientos del primer nivel Max, se calculará el valor de la utilidad para cada movimiento, y sólo para los tres mejores movimientos (aquellos con los mayor utilidad) que generarían tres nuevos posibles estados de la partida, se procederá a analizar desde el segundo nivel Min. En el segundo nivel Min, se procederá de la misma manera, solo los tres posibles estados de la partida generados por los tres mejores movimientos (en este caso aquellos que tengan el menor valor de utilidad), serán los que la función pasará para analizar en el siguiente nivel de profundidad correspondiente al nivel Max. Y así sucesivamente hasta alcanzar el nivel máximo de profundidad. En cuyo caso, tal como se explicó anteriormente se irán devolviendo progresivamente los mejores valores hasta volver nuevamente al principio, es decir primer nivel de profundidad, donde se decide cual es mejor movimiento.

4.4. Adaptación del algoritmo desarrollado al middleware NAOqi

NAOqi es el nombre del software principal que se ejecuta en los robots NAO y que se encarga de controlarlos.

El robot NAO está basado en un procesador Intel Atom de 1.6 GHz que ejecuta un kernel de Linux y un middleware de Aldebaran llamado NAOqi, adecuado para crear nuevas funcionalidades sobre un robot NAO como pueden ser por ejemplo paralelismo, manejo de recursos, sincronización, eventos, etc...

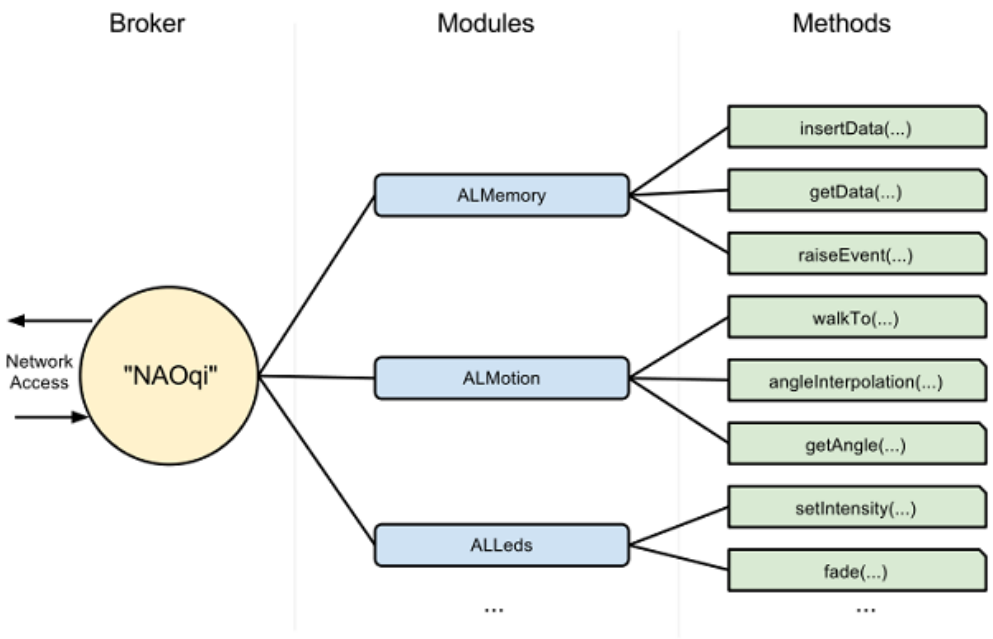
Cuando se inicia el robot NAO, se carga un archivo de preferencias almacenado en el archivo autoload.ini, este archivo define las librerías que debe cargar.



Funcionamiento de autoload.ini

NAOqi se ejecuta en el robot NAO por medio de un bróker. El bróker es un proceso que está escuchando la dirección IP y el puerto, y contiene un conjunto de módulos que ofrecen ciertas funcionalidades a través de funciones de alto de nivel.

El bróker también provee servicios de búsqueda, de modo que cualquier método anunciado por un módulo, puede ser encontrado a través del árbol o de la red. Cargando en forma de árbol los métodos a los módulos, y los módulos a un bróker, como se muestra en la siguiente imagen:



Conexión broker - módulos - métodos

El bróker más importante es, como su propio nombre indica, el MainBroker, el cual contiene los módulos que nos permiten acceder a los sensores y actuadores del robot NAO. Este bróker está escuchando en el puerto 9559.

Típicamente, cada módulo es una clase dentro de una librería. Cuando la librería es cargada con el archivo autoload.ini, automáticamente se instanciarán los módulos. En el constructor de la clase que derivan de "ALModule", se pueden importar "bind" los métodos. Cada módulo contiene varios métodos, algunos de esos métodos pueden ser llamados desde fuera del propio módulo.

Durante el proyecto, hemos utilizado los siguientes módulos que explicaremos con más detalle un poco más adelante:

- **ALMemory**
- **ALSpeechRecognition**
- **ALTextToSpeech**
- **ALLeds**

A pesar de sólo haber utilizado estos cuatro módulos, existen muchísimos módulos más que facilitan el control o ejecución de muchísimas otras funcionalidades como por ejemplo: ALSonar, ALSensors, ALLaser, ALFaceDetection, ALMotion, etc...

Una vez aclarado el funcionamiento del framework y las necesidades de nuestro proyecto, tan sólo hay que añadir las librerías, funciones y métodos que necesitamos en nuestro script.

Lo primero es por tanto importar la librería de NAOqi, esto en Python se debe hacer al principio del script con la siguiente orden:

```
from naoqi import ALProxy
```

Otras cosa que vamos a necesitar serán la dirección IP del robot y el puerto utilizado, para ello, nuevamente al principio del script, se pueden crear dos variables globales que utilizaremos varias veces en nuestro programa, algo como:

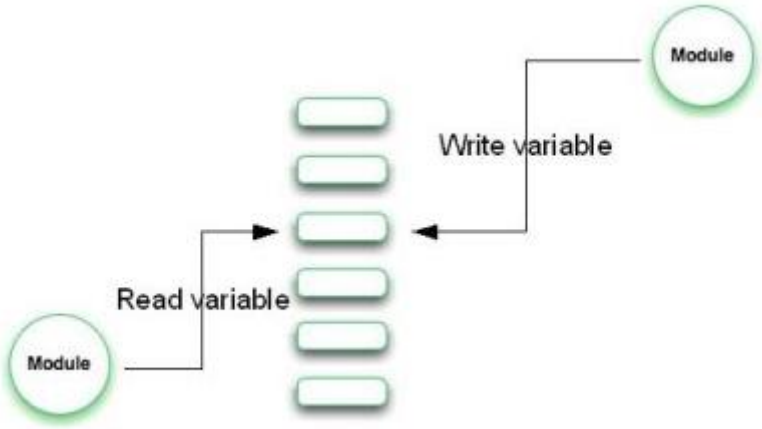
```
IP = "<IP ADDRESS>"

PUERTO = xxxx
```

Por último remarcar un posible problema, NAOqi no es compatible con la última versión de Python. Por tanto conviene empezar cualquier proyecto utilizando la versión 2.7 de Python.

4.4.1. ALMemory

ALMemory es la memoria del robot. Todos los módulos pueden leer o escribir datos, pero hay que tener en cuenta que ALMemory no se sincroniza en tiempo real. ALMemory es un array de valores. ALMemory contiene tres tipos diferentes de datos y provee tres APIs diferentes.



Acceso al módulo ALMemory

4.4.2. ALTextToSpeech

El módulo ALTextToSpeech lo utilizaremos para añadir la funcionalidad del lenguaje en el robot, puesto que se utilizará la voz como medio de comunicación entre el robot NAO y su adversario.

Es decir, para dar constancia al adversario de la ficha que el robot va a mover, o para preguntarle al adversario que ficha va a mover y hasta donde, para tener cierto grado de interacción con el adversario (que se queje cuando le capturen alguna ficha, o que exprese su alegría cuando el mismo le capture una ficha al adversario), así como para expresarse al final de una partida, independientemente de que el tipo de expresión varíe en función de si la ganó, la perdió o quedaron en tablas; implementaremos la voz del robot, para ello, para conseguir que el robot NAO hable, habrá que importar el módulo ALTextToSpeech.

ALTextToSpeech es el módulo que permite al robot hablar a través de un motor de texto y voz (text-to-speech). El resultado de la síntesis se envía a los altavoces del robot NAO.

ALTextToSpeech está basado en un sintetizador del habla proporcionado por "ACAPELA", el cual permite la personalización de voz la mediante "tags" incluidos dentro de la cadena de comando enviada al motor.

La corriente de salida de audio también se puede modificar mediante el uso de uno de los siguientes efectos:

- Pitch shifting: Efecto que modifica el tono inicial de voz
- Double voice: Efecto que produce un efecto de eco en la voz

En el programa desarrollado se necesita crear el objeto que permitirá hablar al robot, y ello lo conseguiremos con la siguiente instrucción:

```
tts = ALProxy("ALTextToSpeech", IP, PUERTO)
```

Posteriormente, estableceremos el lenguaje en español:

```
tts.setLanguage("Spanish")
```

Finalmente el método que utilizaremos del módulo ALTextToSpeech será el método "say", que será el método que se invocara cada vez que el robot tenga que hablar algo, y a este método se le invoca de la siguiente manera:

```
tts.say("Lo que quieras decir")
```

4.4.3. ALSpeechRecognition

El módulo ALSpeechRecognition lo utilizaremos para añadir la funcionalidad del reconocimiento del habla en el robot.

Para tener constancia de los movimientos realizados por el adversario, el robot requerirá conocer el movimiento que realiza el adversario, y esto se conseguirá porque el adversario debe de decir verbalmente la posición de la ficha que va a mover y hasta que posición la va a mover. Para ello necesitamos implementar el reconocimiento de voz en el robot, para que sea capaz de entender el movimiento que ha descrito su adversario y pueda actualizar en su memoria el nuevo estado de la partida.

Para que el robot NAO sea capaz de reconocer las palabras o las frases, habrá que importar el módulo ALSpeechRecognition.

ALSpeechRecognition: es el módulo que da al robot la capacidad de reconocer palabras o frases predefinidas en varios idiomas, en nuestro caso es el módulo utilizado para el reconocimiento del habla del movimiento del oponente.

El módulo funciona de la siguiente forma:

> (1) Antes de empezar el reconocimiento de voz, ALSpeechRecognition necesita haber inicializado una lista de posibles frases a reconocer.
> (2) Una vez empezado, ALSpeechRecognition, utiliza la variable booleana SpeechDetected para comprobar si se ha escuchado algo a través del altavoz.
> (3) Si algo ha sido escuchado, el elemento de la lista (1) (la lista que contiene las posibles palabras o frases a reconocer por el robot NAO) que mejor cuadre con lo que ha sido escuchado por el robot, es almacenado en la variable WordRecognized.

La variable WordRecognised se organiza como sigue:

```
[frase1, probabilidad1, frase2, probabilidad2, ……., 
fraseN, probabilidadN]
```

Donde:

> `fraseI` es una de las frases predefinidas al principio en la lista.
> `probabilidadI` es una estimación de la probabilidad de que sea esa frase lo mismo que ha sido pronunciado por el humano.

Siendo el primer elemento del array, es decir, la `frase1`, la frase con mayor probabilidad. Cabe resaltar que el módulo ALSpeechRecognition no puede ser probado en un robot virtual. En el programa desarrollado para poder utilizar la función de reconocimiento del habla, se necesitará crear el objeto que permita utilizar dicha funcionalidad, esto lo conseguimos a través de la siguiente instrucción:

```
asr = ALProxy("ALSpeechRecognition", IP, PUERTO)
```

Posteriormente, al igual que con ALTextToSpeech, se establece el lenguaje en español:

```
asr.setLanguage("Spanish")
```

A continuación se debe inicializar una lista que contenga las posibles frases a reconocer. En nuestro caso en concreto la lista debe de contener las diferentes coordenadas posibles, para seguidamente asignarle la lista al objeto creado encargado del reconocimiento del habla, quedando algo como:

```
vocabulario = ["coord1", "coord2", …. , coordN]

asr.setVocabulary(vocabulario, False)
```

Donde `False` significa que el robot espera escuchar una de las palabras o frases especificadas, nada más y nada menos. Si por el contrario le asignamos el valor `True`, significará que la palabra o frase puede estar en el medio de otras palabras o frases pronunciadas, entonces el robot intentará detectar alguna de esas palabras o frases en el conjunto de todo el contenido verbal escuchado en un momento dado.

Por otra parte, como vamos a hacer uso de la memoria del robot, recordemos que es el módulo ALMemory, también deberemos de crear dicho objeto con la siguiente instrucción:

```
Memory = ALProxy("ALMemory", IP, PUERTO)
```

Llegado a este momento, y con todo inicializado y listo para usarse, ya podemos utilizar la funcionalidad del reconocimiento del habla. Para ello, cuando se quiera que el robot espere alguna palabra o frase, habrá que llamar a la función de reconocimiento del habla, almacenar, o mejor dicho recuperar aquello que el robot ha escuchado que estará almacenado en la variable WordRecognized del módulo ALMemory, y finalmente llamar a la función para que deje de escuchar y siga haciendo otras cosas. Esto se resume en las siguientes 4 instrucciones:

```
asr.subscribe("Test_ASR")

time.sleep(15)        #El tiempo que deseamos que espere

loqEscucho = memory.getData("WordRecognized")[0] #Para
obtener la palabra con mayor probabilidad de ser la palabra o
frase dicha por el adversario

asr.unsubscribe("Test_ASR")
```

Por tanto una vez inicializados todos los objetos y llamados todos los módulos necesarios, cada vez que necesitemos hacer uso de la función de reconocimiento del habla, bastará con ejecutar estas cuatro instrucciones.

4.4.4. ALLeds

El módulo ALLeds será utilizado para añadir la funcionalidad de los leds en el robot.

Finalmente, para dar la sensación de mayor interacción con el robot, de manera que al menos parezca que se entienda mejor al robot o que nos parezca más familiar al tener constancia de cuando el robot esta "pensando" (calculando su mejor jugada), o incluso, simplemente por hacerlo un poco más divertido o entretenido, se ha valorado la opción de encender o apagar ciertos leds del robot mientras éste está calculando su jugada (pensando).

Para manejar los leds del robot, habrá que importar el módulo ALLeds.

ALLeds es el módulo que permite controlar las luces led del robot NAO. Podemos encontrar luces led en las orejas, la parte táctil de la cabeza, los ojos y los pies de los robots NAO.

A través del módulo ALLeds se puede fácilmente controlar la intensidad de una luz o grupo de luces led (entre 0 y 100%). ALLeds también proporciona animaciones con las luces y métodos muy prácticos para dirigir los grupos o las listas de luces led.

Los colores de las luces led de los ojos y los pies son 'Fullcolour´, lo que significa que se pueden combinar con los tres colores primarios (rojo, verde y azul), pero las luces de las orejas y la parte táctil de la cabeza sólo pueden ser azules.

Al igual que en los módulos anteriores para poder hacer uso del control de los leds de los robots NAO, habrá que previamente inicializar el objeto que hace referencia al módulo correspondiente, en el caso del control de los leds, esto lo conseguiremos a través de:

```
Leds = ALProxy("ALLeds", IP, PUERTO)
```

En nuestro caso vamos a utilizar los leds de la cara, para ello existe un nombre corto reconocible por NAOqi. Por tanto llamaremos a este conjunto de luces como nombreLuces:

```
nombreLuces = 'FaceLeds'
```

Aunque se puede hacer referencia a cualquiera de las luces leds del robot NAO.

Una vez hecho esto, son diferentes las opciones que se pueden utilizar, como por ejemplo simplemente encender y apagar cierto grupo luces con las siguientes instrucciones:

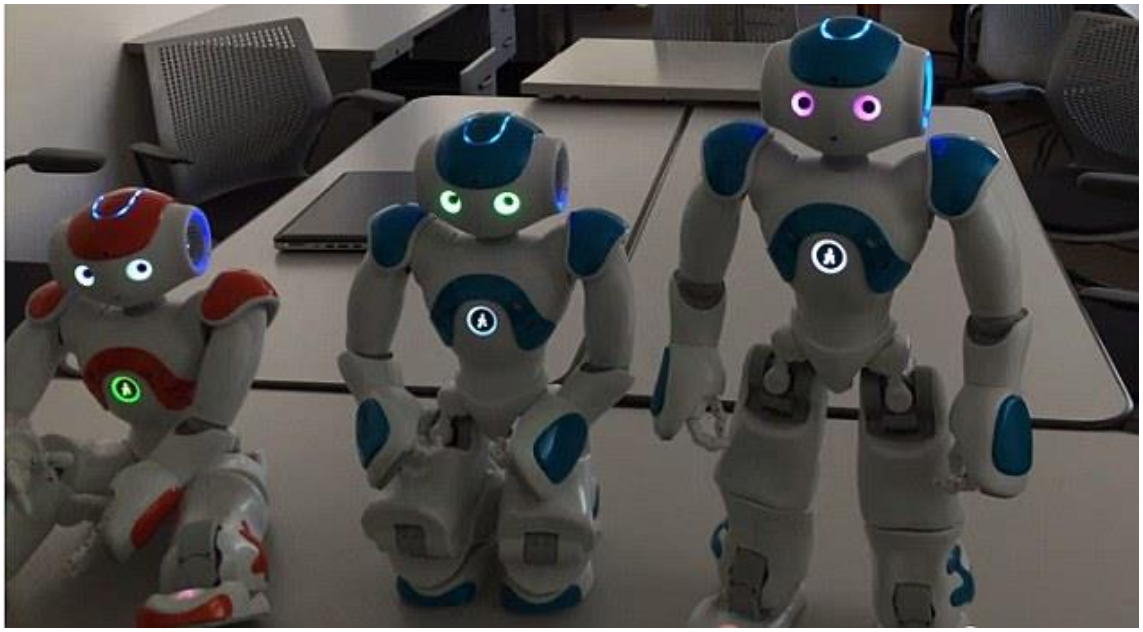
```
Leds.on(nombreLuces)

Leds.off(nombreLuces)
```

O crear un grupo de luces para luego, en un momento determinado de ejecución del programa cambiarles la intensidad:

```
Leds.setIntensity(nombreLuces, 0.5)
```

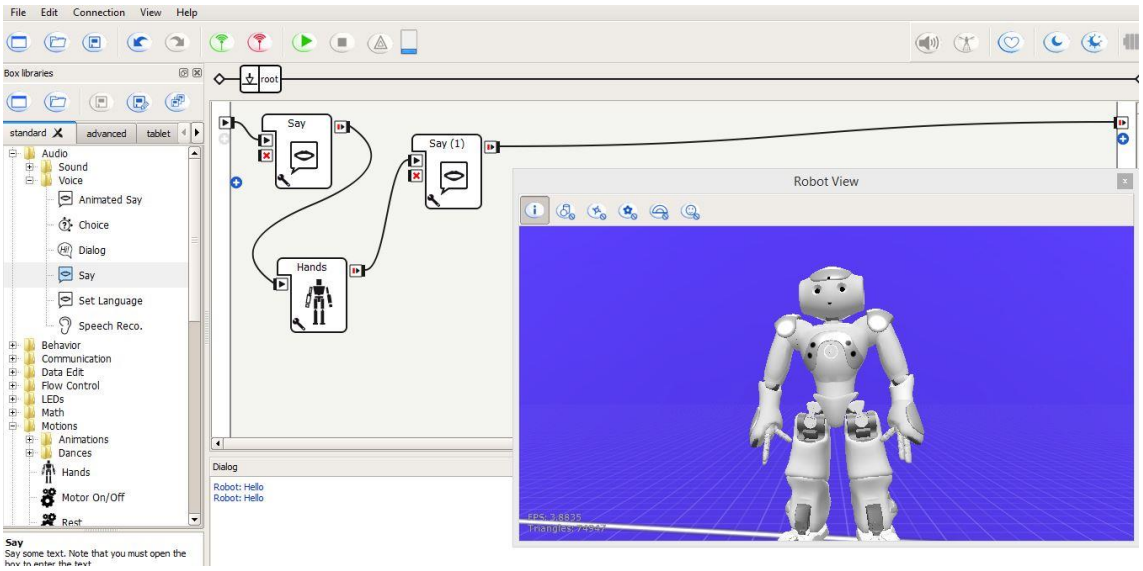
En el programa desarrollado, con la finalidad de que sea más visible, lo que aremos será encender los leds de la cara. Para ello los encenderemos mientras el robot está calculando su jugada. Puesto que tenemos la posibilidad de elegir el color de las luces leds, dado que suele relacionarse el color verde con objetivos conseguidos, y este no es el caso porque el robot NAO tan sólo estará calculando su jugada, y el color rojo suele relacionarse con peligro, y tampoco es el caso, finalmente elegiremos el color azul de los leds para que enciendan mientras el robot NAO esté calculando su próxima jugada o movimiento.



Robots NAO con distintos colores leds

4.5. Utilización de Choregraphe

Esta herramienta proporcionada por Aldebaran ha sido de gran ayuda durante la implementación y testeo del proyecto ya que permite realizar pruebas con el robot virtual obteniendo retroalimentación de forma muy precisa. Por tanto se ha permitido testear al algoritmo progresivamente, excepto como se dijo con anterioridad, con el módulo de ALSpeechRecognition puesto que no se puede ejecutar con el robot virtual disponible en el Choregraphe, sino que se necesita conectarse un robot real para poder ser probado.



Choregraphe conectado al robot virtual

Llegados a este punto, ya es sobradamente conocido que Choregraphe se puede conectar tanto a un robot NAO real como a un robot virtual, para ello, es suficiente con clickar en la pestaña de "Connection" y elegir el dispositivo al que se desee conectar, una vez conectado al dispositivo deseado dándole click al Play o Run, empezará a ejecutarse el programa en el robot permitiéndonos ver los resultados de nuestra programación, si estamos conectados a un robot real NAO, el robot real ejecutará las instrucciones, si por el contrario estamos conectados al robot virtual, podremos ver los resultados en un robot que aparece en la pantalla como el que se muestra en la imagen anterior, además esta ventana se puede redimensionar fácilmente, por lo que se puede adaptar el tamaño de la ventana a nuestras necesidades o preferencias. Además, aunque el robot virtual no habla, tal y como se puede ver en la imagen anterior, en la parte inferior de la imagen, hay una sección llamada "Dialog" que está cubierta parcialmente por la ventana donde se visualiza el robot, en esta sección Dialog se irán escribiendo las frases o palabras que se supone que debería de estar pronunciando nuestro robot virtual.

Todas estas características han permitido poder probar el programa con cada nueva inserción de código implementado para validar el correcto funcionamiento y modificarlo en caso necesario.

En cuanto a la creación de programas utilizando el software de simulación Choregraphe, es relativamente sencillo. Se pueden crear programas de formas distintas, utilizando los fragmentos de código correspondientes al módulo deseado o la funcionalidad que deseamos añadirle al robot, o se pueden creer directamente scripts enteros añadiendo la opción "add box".

Tanto añadiendo determinadas características poco a poco como añadiendo la box para programar nuestro propio script, en el momento en que añadimos algo, aparecerá un cuadro correspondiente a cada función agregada en la sección central (como los 3 cuadros mostrados en la imagen anterior: Say, Say(1) y Hands) de Choregraphe. Estos cuadros disponen de entradas y salidas, dependiendo de la funcionalidad de cada

cuadro, dicho cuadro dispondrá de 1 o más entradas y de 1 o más salidas. Haciendo click desde la salida de un cuadro y arrastrando el click hasta la entrada de otro cuadro, conseguiremos que al finalizar la ejecución del primer cuadro, el programa continúe con la ejecución del segundo cuadro o funcionalidad.

Además para conseguir que el programa entero se ejecute, habrá que conectar la entrada del primer cuadro que queremos que ejecute a la entrada del programa. La entrada del programa en Choregraphe la podemos encontrar en la parte izquierda superior de esta sección del programa donde se encuentran los cuadros (en la imagen anterior se puede apreciar ya que dicha entrada está conectada a la entrada del cuadro Say). Además también habrá que conectar la salida del último cuadro que se debe ejecutar a la salida del programa, indicando de este modo, que al finalizar la ejecución de dicho cuadro, el programa debe terminar. La salida del programa se puede encontrar en la parte derecha superior, también en la misma sección donde están los cuadros (en la imagen anterior también se puede apreciar porque la salida del cuadro Say(1) está conectada a dicha salida).

Otra opción de que dispone Choregraphe, es que una vez añadido el cuadro o los cuadros con las funcionalidades deseadas, haciendo doble click sobre el cuadro, se mostrará una ventana que contiene el código de dicho cuadro previamente establecido por Choregraphe (se remarca y se recuerda que el código utilizado por Choregraphe es Python), permitiéndonos de esta forma poder editar todos los fragmentos de código que deseemos, adaptándolos a nuestras necesidades.

Por ultimo mencionar que los cuadros, contienen previamente implementadas una serie de funciones básicas, como son la funcionalidad en sí que debe realizar el robot cuando le llega la señal de entrada, y las posibles reacciones o mensajes de erros ante situaciones de errores.

Por tanto, crear programas utilizando Choregraphe ha sido tan fácil como añadir la o las “box” con nuestro propio código, y/o los cuadros con las funcionalidades predefinidas disponibles en Choregraphe, editar el código de los cuadros para adaptarlos a nuestras necesidades, y conectar las entradas y salidas correspondientes.

Una vez creado el programa y conectados a un robot NAO real o conectados al robot virtual de Choregraphe, simplemente dándole al botón de “Play” se ejecuta el código del programa mostrándonos los resultados de la ejecución de la totalidad del código sobre un robot NAO.

5. Validación

5.1. Algoritmo

Se han realizado dos tipos distintos de pruebas con el algoritmo, por una parte, antes de integrar el algoritmo al robot se realizaron pruebas de testeo tanto pruebas de caja blanca como pruebas de caja de negra, para finalmente, una vez validados todos los posibles estados y comprobado que el algoritmo funcionaba bien, pasar a realizar pruebas jugando partidas completas.

El primer tipo de pruebas, a su vez, se subdivide en dos tipos distintos de pruebas más, por un lado las pruebas de caja blanca, y por otro lado las pruebas de caja negra.

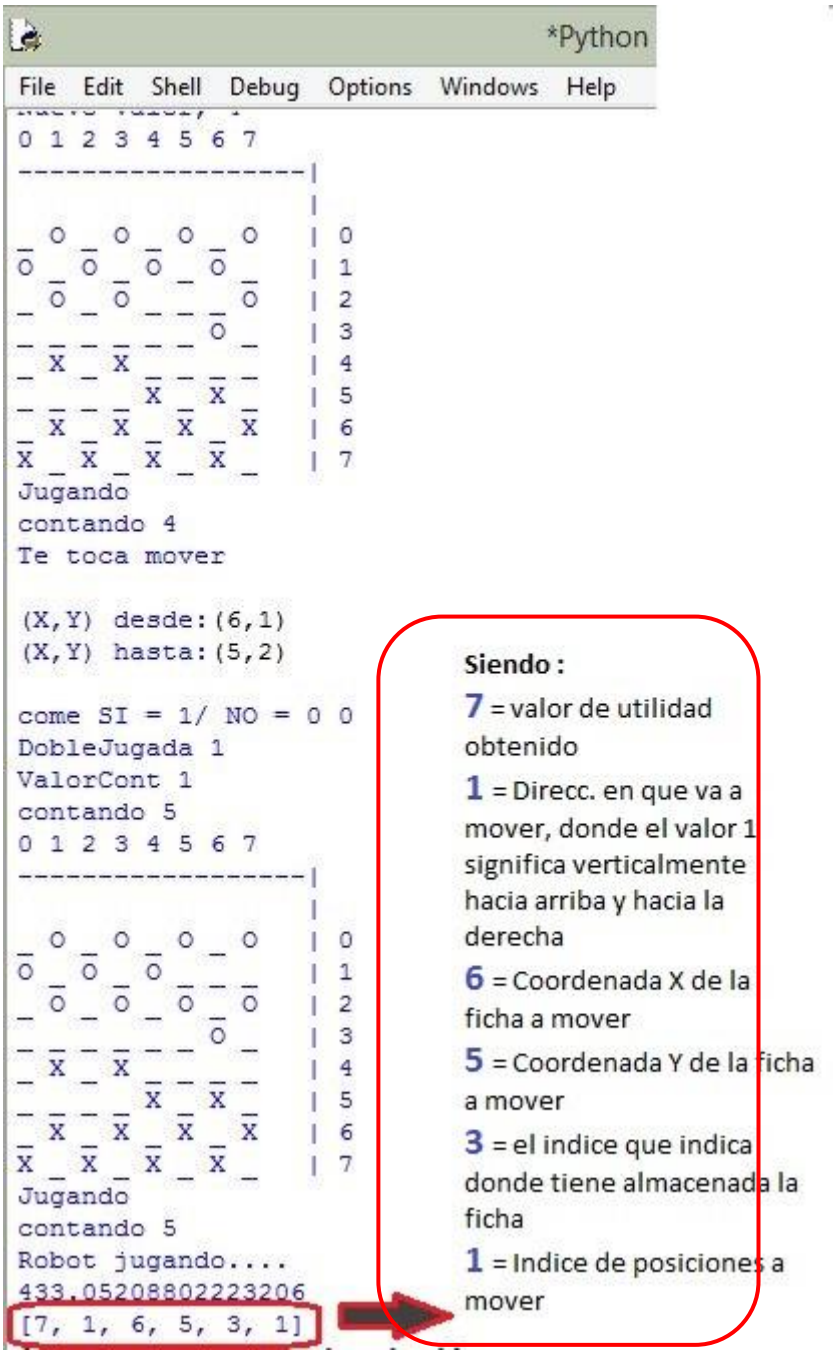
En cuanto a las pruebas de caja blanca, se han realizado para comprobar que las líneas específicas del código funcionaban tal y como estaba previsto, utilizando para ello pruebas de control de flujo, pruebas de bifurcación o branch testing, y pruebas de caminos básicos, todas ellas para comprobar si las definiciones y usos de las variables del script funcionaban correctamente, para poder definir si los bucles o las líneas de código donde existe alguna condición son la mejor opción empleada o deberían de ser cambiadas, para probar las expresiones aritmético-lógicas, y además pruebas para recorrer todos los posibles caminos de ejecución.

En cuanto a las pruebas de caja negra, en base a la intuición y la experiencia se han desarrollado varios estados distintos de la partida teniendo en cuenta tanto situaciones normales sin mayor complejidad como situaciones límite, situaciones de excepción, o situaciones realmente complejas de distintos posibles estados de la partida, consiguiendo con ello, que partiendo de un conjunto de datos de entrada, nos fuesen devueltos los valores de salida proporcionados por el algoritmo, comprobando así el correcto o no funcionamiento del script en situaciones complejas o comprometidas de una partida.

Tanto las pruebas de caja blanca como las pruebas de caja negra han sido realizadas continuamente a lo largo del desarrollo del algoritmo, lo que ha permitido poder ir corrigiendo los errores prácticamente a diario e ir descubriendo nuevas situaciones límite que inicialmente no habían sido tomadas en consideración.

En cuanto al segundo tipo de pruebas, se han realizado aproximadamente 80 pruebas jugando partidas completas de damas a través de la consola de Python, antes de integrar el algoritmo al robot. Para ello se creó una especie de tablero de damas con rayas horizontales separadas por espacios en blanco para simular los cuadros del tablero, donde "X" era utilizado para las fichas de un jugador, y "O" para las fichas de otro jugador. Para este tipo de testeo, las jugadas realizadas por el oponente eran insertadas a través inputs de Strings en el sistema, para posteriormente ser tratados por el script y poder utilizar el nuevo estado de la partida para calcular el mejor

movimiento posible. Por otra parte, tras calcular el mejor movimiento posible, se mostraba por consola los mejores valores de utilidad obtenidos para cada posible dirección de movimientos (es decir verticalmente hacia arriba y hacia abajo tanto hacia la derecha como hacia la izquierda), y también la decisión a la que había llegado el algoritmo, es decir que ficha iba a mover, en qué dirección y cuantas posiciones (ya que en caso de ser una ficha reina, podría mover más de una posición). Además, para no perder de vista el tablero, en cada nueva jugada, tanto la realizada por el algoritmo, como la realizada por el adversario (en este caso yo misma), se volvía a dibujar el nuevo estado de la partida por consola. Además, también se le añadieron un par de instrucciones para que calculase el tiempo empleado en el cálculo de cada movimiento, para así poder realizar un estudio en base a la eficiencia y al tiempo de cómputo empleado por el algoritmo en distintas situaciones de partidas.



Algunos de los errores descubiertos mediante el conjunto de estas pruebas fue por ejemplo, el hecho de que aunque se tenía en consideración que si se atrapaba una ficha, el jugador que había atrapado la ficha volvía a mover nuevamente (hablando de jugadas realizadas), a la hora de calcular el mejor movimiento, durante el algoritmo Minimax, este no tenía en cuenta ese hecho para seguir buscando el mejor camino. Por mencionar otro posible error, gracias a estas pruebas se pudo comprobar también que inicialmente el algoritmo Minimax buscaba los mejores movimientos incluyendo fichas que ya habían sido capturadas y por tanto que ya no se podía jugar con ellas. Otros errores de menor importancia también fueron descubiertos y depurados gracias a estas pruebas.

5.1.2. Pruebas de eficiencia y tiempo de cómputo

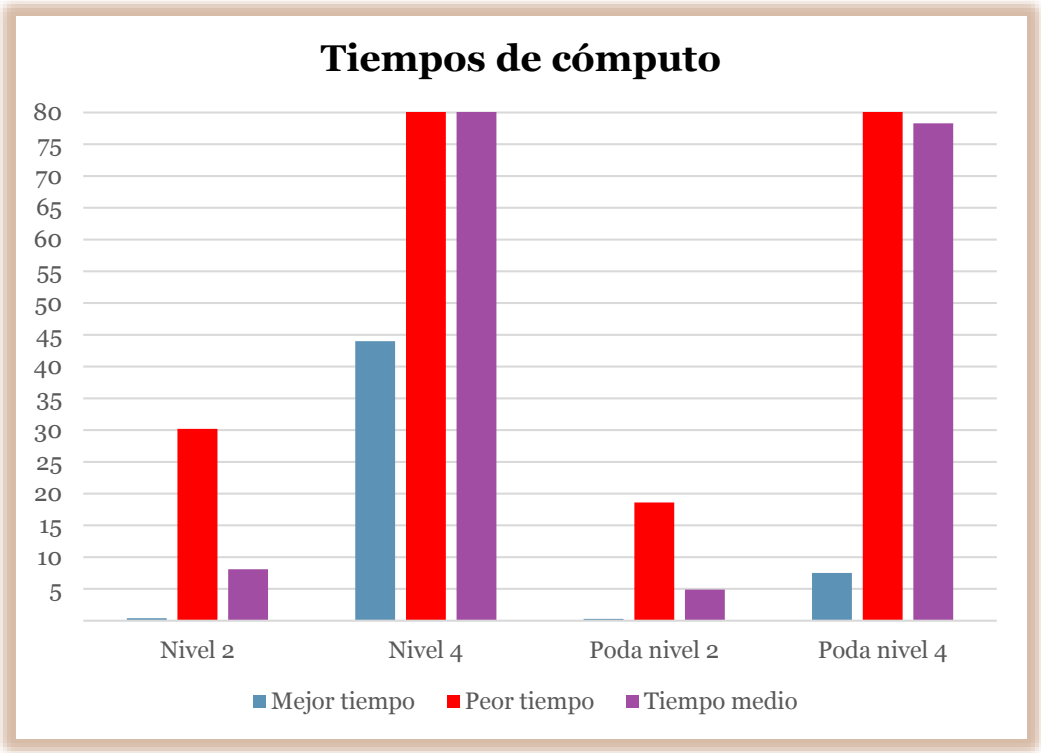
Poder evaluar y/o comparar el tiempo de computo con distintas versiones, es decir con distintas profundidades o aplicando la poda alfa-beta, es indiscutiblemente una tarea difícil debido a la cantidad de variables y distintas situaciones que se pueden dar a lo largo de una partida (número de fichas no bloqueadas tanto del robot como del adversario, teniendo en cuenta que si son reinas tienen más libertad de movimientos y por tanto necesitan más tiempo de computo, y las diferentes direcciones en que se pueden desplazar, es decir si están totalmente bloqueadas, parcialmente bloqueadas, o totalmente libres), y que hacen variar considerablemente el tiempo de computo, cosa que además es más notable a mayor profundidad, pues la diferencia de computo es más grande con pequeñas variaciones.

A pesar de ello, teniendo en cuenta el tiempo de cómputo registrado a lo largo de las aproximadamente 80 partidas completas jugadas a través de la terminal de Python, se han recogido los siguientes datos referentes al menor y al mayor tiempo de cálculo empleado para cada profundidad, así como el tiempo medio de cálculo necesitado:

	Nivel 2	**Nivel 2 con poda**	**Nivel 4**	**Nivel 4 con poda**
Mejor tiempo	0.39s	0.28s	44s	7.5s
Tiempo medio	8.1s	4.9s	4077.5s	78.3s
Peor tiempo	30.2s	18.6s	21517.4s	433.05s

No se añaden los datos obtenidos en niveles de profundidad superiores, puesto que debido a su largo tiempo de ejecución no se han recogido muchos datos.

En la siguiente gráfica se puede apreciar mejor, la gran diferencia de tiempo de cómputo en las distintas profundidades en las que se ha testeado el algoritmo:



5.2. Simulación

Al empezar con Choregraphe, lo primero que hice, fue realizar pruebas con los módulos que se iban a utilizar en nuestro script con la finalidad de familiarizarme con ellos, descubriendo, que lo que aparentemente parecía fácil no lo era tanto, y descubriendo que no podía testear con ciertos módulos simplemente a través del robot virtual del que dispone Choregraphe.

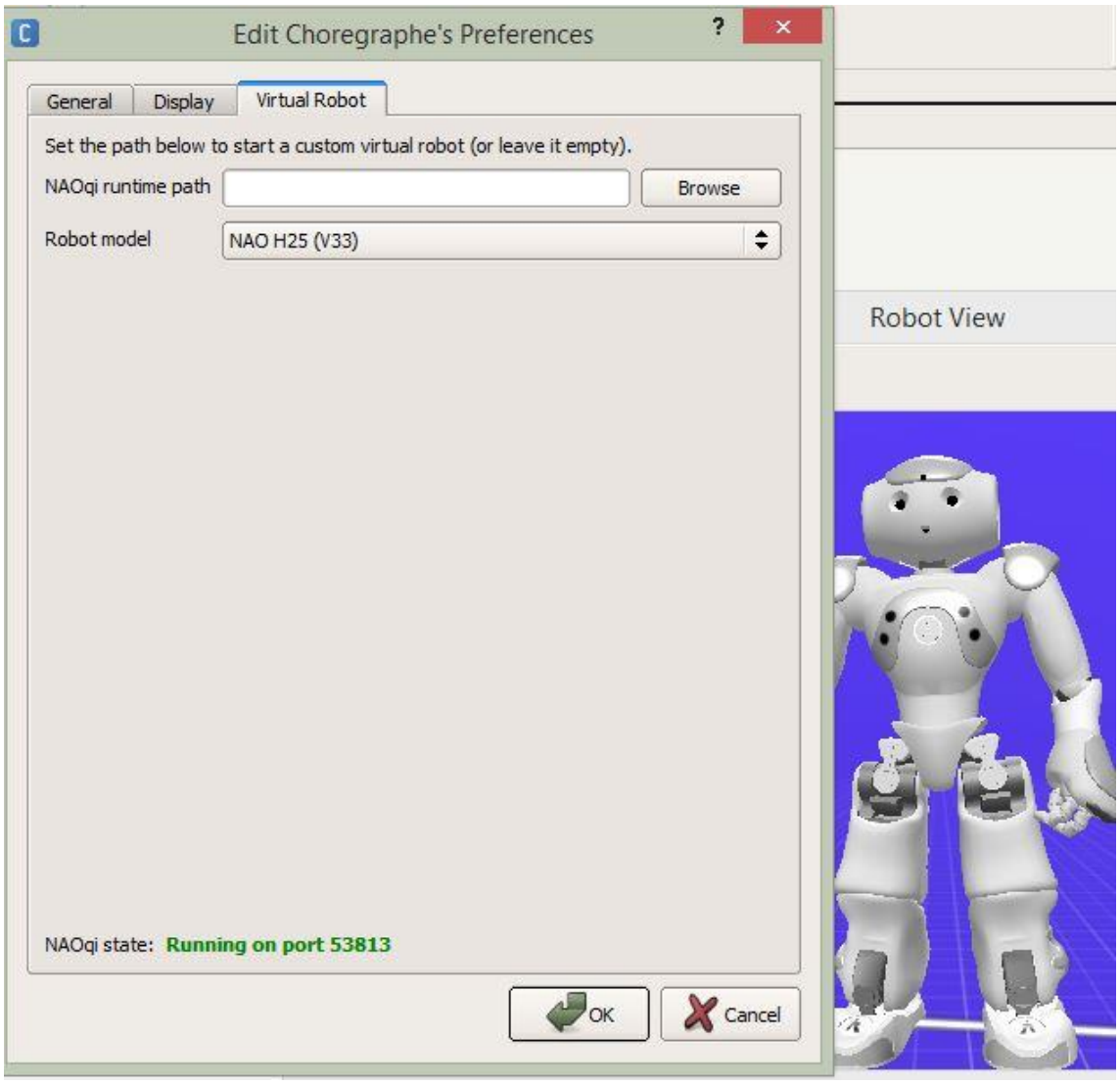
Pasado la fase de familiarización, se procedió a insertar los nuevos segmentos de código relacionados con el habla del robot para poder testear el conjunto en el entorno virtual de Choregraphe, y he aquí el primer problema que encontré, pues Choregraphe está preparado para poder insertar módulos o partes de códigos con ciertas funcionalidades de forma bastante guiada, e ir conectándolos luego uno tras otro, justo por este motivo me fue bastante difícil encontrar la forma de insertar mi script en Choregraphe. Finalmente se descubrió que al insertar las librerías de NAOqi, si se especificaba como dirección IP el localhost, y como puerto el puerto en donde estaba escuchando el robot virtual de Choregraphe, al ejecutarlo de forma normal en la consola de Python, automáticamente las ordenes llegaban al robot virtual de Choregraphe y este ejecutaba el script.

Una vez superado el primer problema con Choregraphe se lanzó el script con el robot virtual de manera que el robot escribía en el cuadro “Dialog” de Choregraphe lo

que se suponía que iba diciendo porque en realidad, el robot virtual tampoco puede hablar.

Después de implementar la funcionalidad del habla se intentó implementar la funcionalidad del reconocimiento del habla para capturar los movimientos de su adversario, encontrándonos con el segundo problema, imposible de implementar y/o ejecutar utilizando Choregraphe. Por ello se implementó una posible primera versión a falta de ser probada con el robot NAO real.

Finalmente, se intentó implementar la funcionalidad de los leds, descubriendo q a pesar de no haber ningún comentario al respecto, y de no saltar ningún error con Choregraphe (cosa que si ocurría al intentar implementar la funcionalidad del reconocimiento del habla), el script se ejecuta correctamente pero no se encendía ningún led, llegando finalmente a la conclusión, que para probar la funcionalidad de los leds, también iba a ser necesario utilizar el robot NAO real.



Encontrar en puerto para el robot virtual en Choregraphe

Como consecuencia de la dificultad o imposibilidad descubierta de probar ciertos módulos sobre un robot virtual, entre las pruebas realizas previas a dicho descubrimiento, y las pruebas realizas después de conocer dichas restricciones, se realizaron más de cien pruebas con el robot virtual a través de Choregraphe, no todas ellas fueron partidas completas de damas, si no que en esta ocasión un alto porcentaje de las pruebas fueron orientadas a probar las funcionalidades del robot, aunque también se realizaron pruebas de partidas completas a través de Choregraphe utilizando simultáneamente la consola de Python para poder insertarle al robot virtual los movimientos de su adversario (yo misma), debido a la imposibilidad de poder desarrollar la partida con el reconocimiento del habla; En esta fase del proyecto, también se jugaron partidas del robot virtual contra familiares y amigos para tener mayor seguridad a cerca de su robustez.

5.3. Pruebas físicas

Para las pruebas físicas se utilizó uno de los robots de los que dispone el departamento, al correr por primera vez el script para testearlo, en seguida me di cuenta que había que encontrar una forma más rápida de poder probarlo, así que realice una serie de scripts para probar cada una de las funcionalidades que deseaba, es decir, un script para probar el habla del robot, otro script para probar el reconocimiento del habla, otro script para probar distintas ejecuciones con los leds, y finalmente otro script que incluyera las tres funcionalidades.

Estos scripts variaron a lo largo de las diferentes pruebas.

Al probar con el script del habla fue cuando se detectó que todas las frases tenían el mismo tono o énfasis de voz que quedaba un poco monótono, y fue entonces cuando se decidió variar un poco el tono de voz en la medida de lo posible para distintas frases en distintas situaciones. Además se comprobó que el robot no sabía leer números (equivalentes a las coordenadas) y que por tanto previamente había que convertir los enteros a strings.

Seguidamente se probó con el script de los leds que avanzo sin mayores problemas, simplemente probando distintas funciones.

A continuación se probó el script con funciones de reconocimiento del habla, fue entonces cuando se vio la necesidad de crear un diccionario para convertir las coordenadas percibidas por el robot a través del habla en forma de String a enteros que son los que usan para calcular el movimiento a realizar por el algoritmo. Se advirtió también que para que el robot esperase el tiempo suficiente como para escuchar las coordenadas procedentes de su adversario, había que añadirle también “time.sleep(10)”, y además el tiempo de procesamiento de reconocimiento del habla es mayor del esperado.

Finalmente, a lo largo de las sesiones con el robot NAO real, se realizaron 15 partidas a las damas con el robot, las partidas se desarrollaron con total normalidad, con la única salvedad de que en ocasiones el robot no entendía bien las coordenadas, motivo por el cual se decidió implementar una función de comprobación de forma que

tras escuchar las coordenadas procedentes de su adversario y procesarlas utilizando el módulo ALSpeechRecognition, el robot NAO repite aquello que ha escuchado para a continuación decir “¿Es esto correcto?” y quedarse a la espera de un “Si” o un “No” procedente de su adversario, que de nuevo procesara a través del módulo de reconocimiento del habla, en caso afirmativo continuara con su ejecución, y en caso negativo volverá a preguntar por las coordenadas. Al igual que cuando el adversario intenta decirle un movimiento invalido, también responde que el movimiento no es válido y le pregunta de nuevo por unas coordenadas válidas.

6. Conclusiones

El propósito de este proyecto consistía en desarrollar una aplicación que permitiese a un robot humanoide NAO jugar al juego de las damas, de tal manera que el robot fuese capaz de pensar y calcular su jugada, dando constancia verbalmente a su adversario del movimiento que desea realizar, y capturando las jugadas de su adversario a través del reconocimiento de voz. Para ello ha sido necesario el desarrollo de un algoritmo que permitiese al robot NAO calcular sus jugadas, la integración de dicho algoritmo en el sistema del robot, y los testeos físicos necesarios tanto para fijar ciertas funcionalidades que requieren de la presencia física del robot NAO, como también para depurar ciertos errores ocasionados al poner en marcha todo el conjunto desarrollado durante el proyecto. Siendo necesaria la integración de diferentes tecnologías, y teniendo en cuenta todos los aspectos técnicos.

Con todo esto, se puede dividir el desarrollo del proyecto en tres partes claramente diferenciadas, la creación del algoritmo Minimax para el cálculo de la jugada, la integración del algoritmo desarrollado en el robot a través del software de programación y de simulación NAOqi y Choregraphe proporcionados por la empresa Aldebaran, y finalmente las pruebas, la codificación y los testeos llevados a cabo utilizando un robot NAO real.

En cuanto al desarrollo del algoritmo Minimax, se concluye que un algoritmo que en principio parecía sencillo puede complicarse infinitamente con unas pocas reglas y casos especiales.

Tal y como se explicó en el desarrollo del proyecto, el código del algoritmo se ha desarrollado para poder modificar la profundidad de búsqueda deseada, permitiendo así entre otras cosas poder validarlo con distintas profundidades de búsqueda.

En cuanto a la eficacia, el tiempo medio de cálculo de la jugada de nuestro algoritmo Minimax es del orden de 8 segundos para un nivel 2 de profundidad, lo que es considerablemente menor al orden de 68 minutos obtenidos para un nivel 4 de profundidad. De momento el tiempo de cálculo para profundidades superiores es desorbitado y se desestima. Para compensarlo, e intentar encontrar un buen compromiso entre la rapidez y la calidad de las jugadas, se desarrolló una poda alfa-beta para poder aplicarla al algoritmo Minimax, de la que se concluye que el tiempo medio de computo durante una partida es del orden de 80 segundos para un nivel de profundidad 4, lo cual nos lleva a la conclusión de que un algoritmo Minimax con poda alfa-beta reduce considerablemente el tiempo de computo, y que por tanto, si se lleva a cabo una buena implementación de poda, los resultados o movimientos aplicados serán los mismos que si no se aplicase la poda alfa-beta pero con menos tiempo de cálculo, pero que sin embargo, si no se aplica una buena poda alfa-beta, el tiempo de cálculo seguirá siendo menor, pero podrían perderse algunos buenos movimientos.

Por tanto, tras el desarrollo y la validación, se tiene que la activación de la poda alfa-beta en nuestro algoritmo Minimax, es bastante efectiva y reduce el tiempo de búsqueda en casi un 50% para un nivel 2 de profundidad, y más de un 50% para mayores niveles de profundidad.

En cuanto a la integración del algoritmo en el robot NAO, gracias a las librerías, módulos, y métodos de estos, contenidos en el bróker de NAOqi, ha sido relativamente sencillo. Además, el software de simulación Choregraphe, por lo general ha sido de gran ayuda, permitiendo ver en la mayoría de las situaciones la reacciones del robot NAO, lo que a su vez nos ha dado la oportunidad de probar diferentes situaciones e integrar adecuadamente el código previamente desarrollado sin necesidad de desplazarnos al laboratorio para poder interactuar con un robot NAO real.

Finalmente, la integración del código desarrollado en un robot NAO real ha permitido poder terminar de desarrollar el código con las funciones pertinentes como la del reconocimiento del habla, adaptándolo a las nuevas necesidades que surgieron en dicho momento. Además, nos ha permitido probar el programa en un entorno real, y poder testearlo a través de varias partidas contra el robot NAO, comprobando que al tiempo de computo del algoritmo en si, además hay que añadirle el tiempo de procesamiento del robot y otros delays, como el empleado durante el reconocimiento del habla, o durante la ejecución de sus propias frases, que también tuvieron que ser modificadas levemente para que no sonaran demasiado monótonas y darle un poco más en énfasis en ciertos momentos para ciertas frases, consiguiendo así, una expresión algo más similar a las expresiones humanas.

A la finalización de este proyecto se han conseguido alcanzar todos los objetivos que se habían propuesto. Además, se ha documentado toda la información necesaria y el trabajo realizado, facilitando la futura aplicación o ampliación de algunas funcionalidades.

Finalmente, después de someter a nuestro sistema a prueba, en más de 60 partidas, se ha mostrado robusto y el robot ha ganado en todos los casos.

A pesar de ello existen una serie de mejoras que se le podrían aplicar y que se explican a continuación.

6.1. Mejoras futuras:

En función de los resultados obtenidos en la parte de validación, y de las conclusiones a las que se han llegado, sería muy interesante, poder estudiar más profundamente el algoritmo Minimax para poder modificarlo y/o introducirle mejoras que nos permitan reducir el tiempo de cómputo con la finalidad de conseguir un mayor rendimiento y eficiencia. Siguiendo con el mismo argumento, otra posible mejora consistiría en introducir cambios en la heurística de la poda alfa-beta para poder seguir reduciendo el tiempo de cómputo asegurándonos de no poder ningún "mejor movimiento" durante las podas.

Tal y como se ha hecho hincapié anteriormente, puesto que el algoritmo Minimax está desarrollado de forma que se pueda seleccionar el nivel de profundidad al que se desea llegar durante la búsqueda del mejor movimiento (a mayor profundidad, mayor precisión), una posible mejora consistirá en utilizar dicho valor como una variable variable permitiendo la ejecución de partidas en distintos niveles, de manera que se permita al adversario seleccionar el nivel de dificultad deseado para cada partida, siendo por ejemplo el nivel más fácil o nivel principiante el correspondiente a un valor de profundidad igual a uno.

Por otro lado, existen una serie de mejoras respecto a las funcionalidades del robot NAO que podrían ser integrarlas para conseguir un mayor acercamiento a la realidad humana y que proporcionarían mayor autonomía al robot NAO. Una de ellas consistiría en dotar al robot NAO de la funcionalidad necesaria para poder mover sus fichas, para ello podría utilizarse el módulo NAOqi Motion ya que incorpora una gran variedad de posibilidades en cuanto al movimiento del robot, especialmente ALMotion. Por otra parte, y siguiendo con la misma línea de razonamiento, otra posible mejora consistiría en añadir alguna funcionalidad de visión a través de las cámaras de robot NAO que le permitiese reconocer el tablero, sus fichas, las fichas del adversario y que fuese capaz de detectar visualmente los movimientos realizados por su oponente, para ello se recomienda hacer uso del módulo NAOqi Vision.

Con el propósito de seguir acercando los comportamientos del robot NAO a los comportamientos humanos, otra posible mejora consistiría en detectar los intentos de hacer trampas de sus adversarios y reaccionar ante dichos intentos.

Finalmente, para darle un valor añadido, se podrían implementar otra clase de mejoras como por ejemplo añadir la funcionalidad para que se pueda elegir entre distintas variedades de juegos de damas, o incluso añadir las mejoras, desarrollar otros algoritmos y otras necesidades relacionadas, que permitan al robot NAO ser capaz de jugar a distintos juegos de mesa como ajedrez, dominó, parchís, etc..., dando la posibilidad al adversario de decidir a qué juego de mesa quiere jugar.

7. Bibliografía

1. Apuntes de la asignatura de SIN de tercero curso de grado en Ingenieria informática de la UPV
2. Barrientos, A., Fundamentos de la robotica, Ac-Graw-Hill, 1997
3. “Evolving an expert checkers playing program without using human expertise”, K.Chellapilla, David B. Fogel
4. Groover, M.P, Industrial Robotics, Technology, Programming & Application, Mac-Graw-Hill, 1986
5. "Inteligencia Artificial" Russell & Nurving
6. “Machine learning using a genetic algorithm to optimise a draughts program board evaluation function”, Kenneth J.Chisholm and Peter V.G.Bradbeer
7. Stuart Russell y Peter Norving “Inteligencia Artificial, Un enfoque moderno”
8. [Ref_1] - http://blogs.20minutos.es/madrereciente/2015/08/13/las-ventajas-de-los-juegos-de-mesa-para-los-ninos-tambien-en-vacaciones/
9. https://courses.cs.washington.edu/courses/csep573/11wi/lectures/05-games.pdf
10. http://doc.aldebaran.com/
11. https://docs.python.org/2/tutorial/
12. https://es.wikipedia.org
13. https://fenix.tecnico.ulisboa.pt/downloadFile/395143159112/dissertacao.pdf
14. http://juegos-y-hobbies.practicopedia.lainformacion.com/juegos-de-mesa/como-jugar-a-las-damas-398
15. http://motherboard.vice.com/es
16. http://robotyka.ia.pw.edu.pl/papers/tbem-bsc-09-twiki.pdf
17. http://www.cs.caltech.edu/~vhuang/cs20/c/applet/Checkers.java
18. http://www.dawnofwargame.com/es
19. http://www.omniplex.org
20. https://www.quora.com/How-do-I-program-the-AI-for-a-checkers-game-program
21. http://www.rae.es/

8. Apéndice A
Manual de instalación

En este apéndice se describen los pasos necesarios para instalar los diferentes software y/o herramientas utilizadas durante el proyecto, partiendo de que el sistema operativo utilizado para el desarrollo del proyecto ha sido Windows

A.1. Python

Descargar desde *https://www.python.org/downloads/release/python-2710/* la versión Windows x86-64 MSI installer.

Una vez descargado, hacemos doble click en el instalador y aceptamos todas las condiciones del instalador sin preocuparnos de las configuraciones avanzadas.

Una vez finalizada la instalación, ya tendremos Python corriendo en nuestro sistema disponible para empezar a programar.

A.3. Choregraphe

El instalador del simulador Choregraphe puede obtenerse en la siguiente URL: *https://community.aldebaran.com/en/resources/software*, con permiso para poder utilizarlo durante 90 días, pasado ese periodo se deberá de introducir una licencia para poder seguir usándolo. Pero por otra parte si te registras como desarrollador NAO, se te permite seguir utilizándolo sin licencia.

Una vez descargado el instalador, hacemos doble click y aceptamos de nuevo todas las condiciones del instalador.

Una vez finalizada la instalación ya tendremos disponible en nuestro ordenador el simulador Choregraphe y un programa llamado Monitor que nos permite acceder a las cámaras y a la memoria de nuestro robot NAO.

A.2. NAOqi

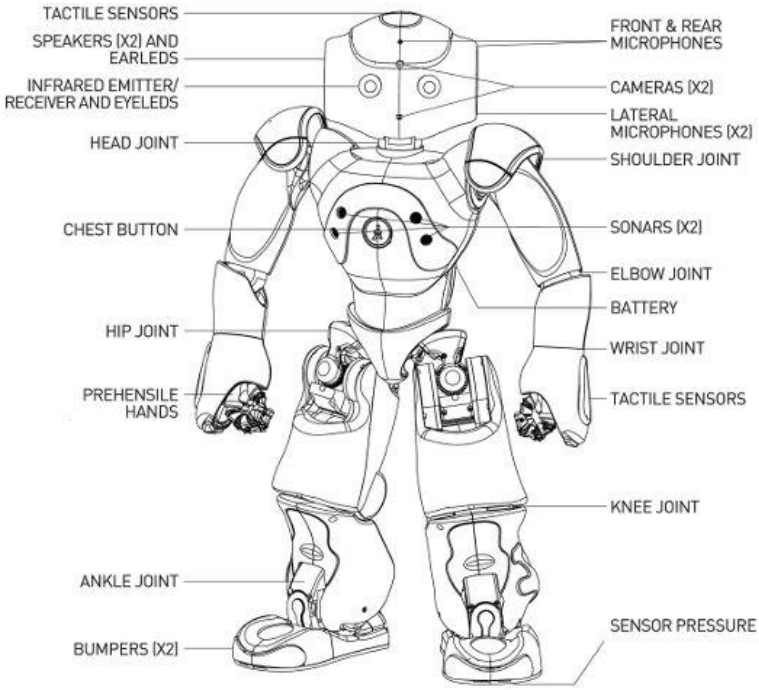
NAOqi también se puede descargar desde la página de Aldebaran (https://community.aldebaran.com/en/resources/software/language/en-gb/robot/nao-2) si te registras como desarrollador.

Antes de su instalación se debe comprobar que la versión de NAOqi sea la adecuada tanto para Python como para la versión descargada de Choregraphe.

Para instalarlo será suficiente con descomprimir el archivo en el directorio en el que va a ser utilizado y hacer doble click. De nuevo el proceso de instalación es muy fácil al ser un proceso guiado.

9. Apéndice B

Especificaciones técnicas del robot NAO



TECHNICAL SPECIFICATIONS

ELECTRICAL

INPUT	100 to 240 Vac - 50/60Hz - Max 1.2A
OUTPUT	25.2 Vdc - 2A

BATTERY		
	Type	Lithium-Ion
	Nominal voltage/capacity	21.6V / 2.15Ah
	Max charge voltage	24.9V
	Recommended charge current	2A
	Max charge/discharge current	3.0A / 2.0A
	Energy	27.6Wh
	Charging duration	5h
	Autonomy	60min (Active use) 90min (Normal use)

CONSTRUCTION

DIMENSION (HxDxW)	573x275x311mm / 22.5x10.8x12.2 inch
WEIGHT	5.2kg / 11.4 lb
CONSTRUCTION MATERIAL	ABS-PC / PA-66 / XCF-30

LANGUAGES

TEXT TO SPEECH & AUTOMATIC SPEECH RECOGNITION	Arabic, Brazilian (Portuguese), Chinese, Czech, Danish, Dutch, English, Finnish, French, German, Italian, Japanese, Korean, Polish, Portuguese, Spanish, Swedish, Russian, Turkish

MOTHER BOARD

CPU PROCESSOR	ATOM Z530	
	Cache memory	512KB
	Clock speed	1.6GHZ
	FSB speed	533mHz
RAM	1GB	
FLASH MEMORY	2GB	
MICRO SDHC	8GB	

CONNECTION

ETHERNET	1×RJ45 - 10/100/1000 BASE T
WIFI	IEEE 802.11b/g/n

AUDIO

LOUD SPEAKERS	×2 lateral	
	Diameter	36mm
	Impedance	8ohms
	Sp level	87dB/w +/- 3dB
	Freq range	up to ~20kHz
	Input	2W
MICROPHONE	×4 on the head	
	Sensitivity	~40 +/-3dB
	Frequency range	20Hz-20kHz
	Signal/noise ratio	58dBA

VISION

CAMERAS	×2 on front	
Sensor model	MT9M114	
Sensor type	SOC Image Sensor	
IMAGING ARRAY	Resolution	1.22MP
	Optical format	1/6inch
	Active Pixels (HxV)	1288×968
SENSITIVITY	Pixel size	1.9µm
	Dynamic range	70dB
	Signal/Noise ratio (max)	37dB
	Responsivity	2.24 V/lux-sec (960p) 8.96 V/lux-sec (VGA)
OUTPUT	Camera output	960p@30fps
	Data Format	YUV422
	Shutter type	ERS (Electronic Rolling Shutter)
VIEW	Field of view	72.6°DFOV (60.9°HFOV, 47.6VFOV)
	Focus range	30cm ~ infinity
	Focus type	Fixed focus

FRAMERATE

Resolution	Embedded	Gigabit Ethernet	100Mb Ethernet	Wifi g
160×120px	30fps	30fps	30fps	30fps
320×240px	30fps	30fps	30fps	11fps
640×480px	30fps	30fps	12fps	2.5fps
1280×960px	29fps	10fps	3fps	0.5fps

Note: using the video stream in remote highly depends on the network and the video resolution chosen. All frame rates depend on the CPU usage. Values are calculated with a CPU fully dedicated to images gathering.

TECHNICAL SPECIFICATIONS

IR

NUMBER	×2 on front
WAVELENGTH	940nm
EMISSION ANGLE	+/-60°
POWER	8mW/sr

SONAR

EMITTERS	×2 on front
RECEIVERS	×2 on front
FREQUENCY	40kHz
SENSITIVITY	-86dB
RESOLUTION	1cm
DETECTION RANGE	0.25m to 2.55m
EFFECTIVE CONE	60°

INERTIAL UNIT

GYROMETER	×2	
	Axis	1 per gyrometer
	Precision	5%
	Angular speed	~500°/s

FSR (FORCE SENSITIVE RESISTORS)

RANGE	0 to 110N
	×4 per feet

ACCELEROMETER	×1	
	Axis	3
	Precision	1%
	Acceleration	~2g

POSITION SENSORS

	NAO HUMANOID	
MRE (Magnetic Rotary Encoder)	×36	
	Using hall effect sensor technology	
	Precision:	12bits / 0.1°

SOFTWARE

OPEN NAO	Embedded GNU/Linux Distribution based on Gentoo
ARCHITECTURE	×86
PROGRAMMING	Embedded: C++ / Python Remote: C++ / Python / .NET / Java / MatLab

LEDS

PLACEMENT	QUANTITY	DESCRIPTION
Tactile Head	×12	16 Blue levels
Eyes	2×8	RGB FullColor
Ears	2×10	16 Blue levels
Chest button	×1	RGB FullColor
Feet	2×1	RGB FullColor

CONTACT SENSOR

	NAO HUMANOID
Chest Button	✓
Foot Bumper	✓
Tactile Head	✓
Tactile Hand	✓

DEGREES OF FREEDOM

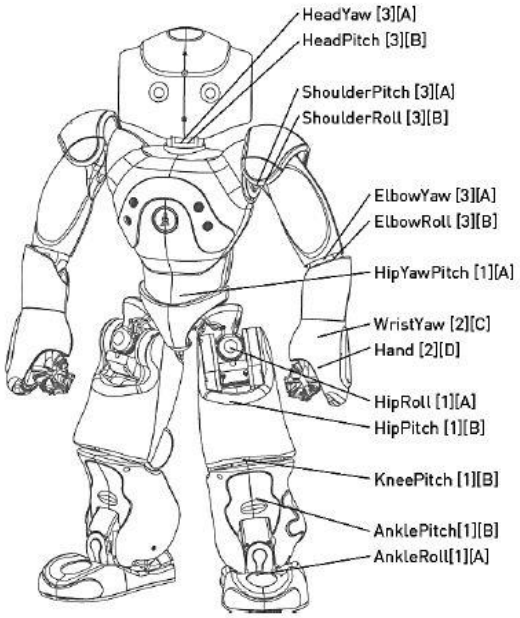
	NAO HUMANOID
HEAD	×2 dof
ARM (IN EACH)	×5 dof
PELVIS	×1 dof
LEG (IN EACH)	×5 dof
HAND (IN EACH)	×1 dof

MOTOR SPECIFICATIONS

MOTOR TYPE Brush DC Coreless

POSITION OF MOTORS

		MOTOR	REDUCTION RATIO
HEAD JOINTS	HeadYaw	Type 3	**Type A**
	HeadPitch	Type 3	**Type B**
ARM JOINTS	ShoulderPitch	Type 3	**Type A**
	ShoulderRoll	Type 3	**Type B**
	ElbowYaw	Type 3	**Type A**
	ElbowRoll	Type 3	**Type B**
	WristYaw	Type 2	**Type C**
	Hand	Type 2	**Type D**
LEG JOINTS	HipYawPitch	Type 1	**Type A**
	HipRoll	Type 1	**Type A**
	HipPitch	Type 1	**Type B**
	KneePitch	Type 1	**Type B**
	AnklePitch	Type 1	**Type B**
	AnkleRoll	Type 1	**Type A**



DESCRIPTION OF THE MOTORS

DESCRIPTION OF THE MOTORS

	MOTOR TYPE 1	MOTOR TYPE 2	MOTOR TYPE 3
Model	22NT82213P	17N88208E	16GT83210E
No load speed	8300rpm ±10%	8400rpm ±12%	10700rpm ±10%
Stall torque	68mNm ±8%	9.4mNm ±8%	14.3mNm ±8%
Continuous torque	16.1mNm max	4.9mNm max	6.2mNm max

Legend: Joint Name[Motor Type][Reductor Type]

SPEED REDUCTION RATIO
TYPE A

	MOTOR TYPE 1	MOTOR TYPE 3
Reduction ratio	201.3	150.27

SPEED REDUCTION RATIO
TYPE B

	MOTOR TYPE 1	MOTOR TYPE 3
Reduction ratio	130.85	173.22

SPEED REDUCTION RATIO
TYPE C

	MOTOR TYPE 2
Reduction ratio	50.61

SPEED REDUCTION RATIO
TYPE D

	MOTOR TYPE 2
Reduction ratio	36.24

CERTIFICATIONS & APPROVALS

REGION	CLASSIFICATION
Europe	CE [Declaration of Conformity]
USA	FCC

ELECTROMAGNETIC COMPATIBILITY	EN 301 489-1 / EN 301 489-17 / EN 300 328 EN 62311 : 2008 / FCC PART15, Class A
SAFETY	IEC 60950-1:2005 (2nd edition)

	MOTOR TYPE 1	MOTOR TYPE 2	MOTOR TYPE 3
Model	22NT82213P	17N88208E	16GT83210E
No load speed	8300rpm ±10%	8400rpm ±12%	10700rpm ±10%
Stall torque	68mNm ±8%	9.4mNm ±8%	14.3mNm ±8%
Continuous torque	16.1mNm max	4.9mNm max	6.2mNm max

Legend: Joint Name[Motor Type][Reductor Type]

SPEED REDUCTION RATIO
TYPE A

	MOTOR TYPE 1	MOTOR TYPE 3
Reduction ratio	201.3	150.27

SPEED REDUCTION RATIO
TYPE B

	MOTOR TYPE 1	MOTOR TYPE 3
Reduction ratio	130.85	173.22

SPEED REDUCTION RATIO
TYPE C

	MOTOR TYPE 2
Reduction ratio	50.61

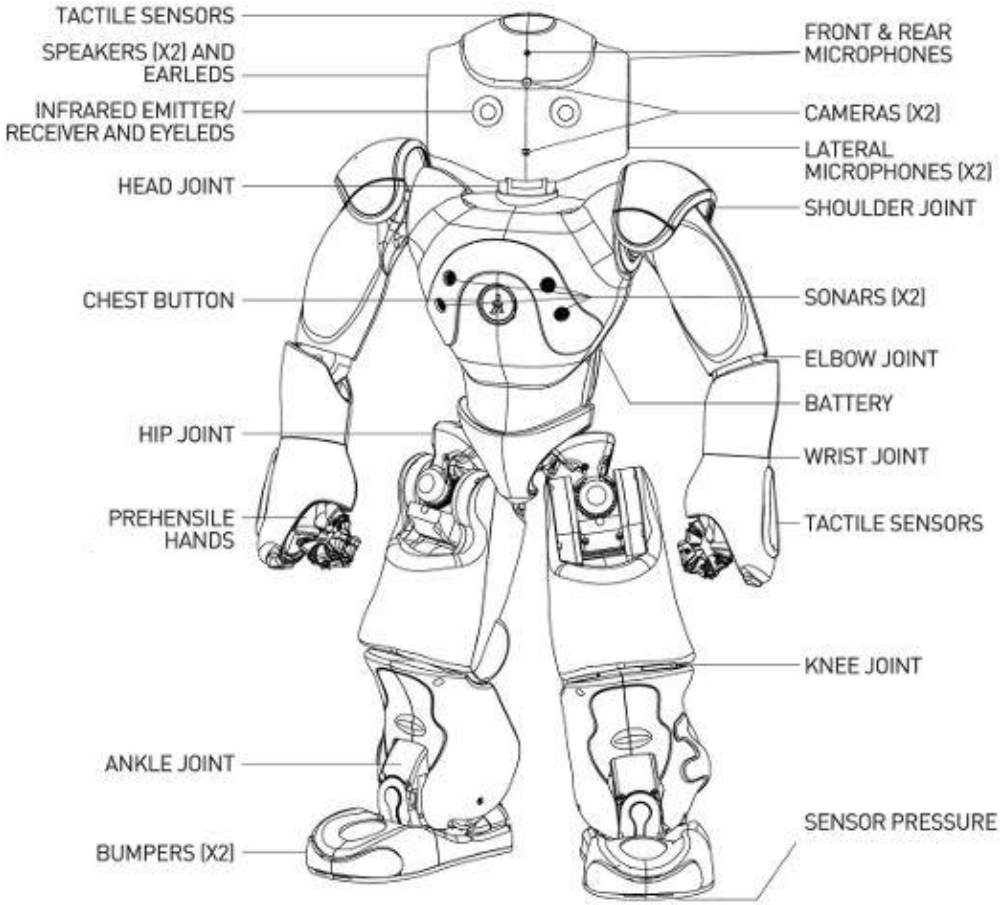
SPEED REDUCTION RATIO
TYPE D

	MOTOR TYPE 2
Reduction ratio	36.24

CERTIFICATIONS & APPROVALS

REGION	CLASSIFICATION
Europe	CE [Declaration of Conformity]
USA	FCC

ELECTROMAGNETIC COMPATIBILITY	EN 301 489-1 / EN 301 489-17 / EN 300 328 EN 62311 : 2008 / FCC PART15, Class A
SAFETY	IEC 60950-1:2005 (2nd edition)


TACTILE SENSORS
SPEAKERS (X2) AND EARLEDS
INFRARED EMITTER/ RECEIVER AND EYELEDS
HEAD JOINT
CHEST BUTTON
HIP JOINT
PREHENSILE HANDS
ANKLE JOINT
BUMPERS (X2)
FRONT & REAR MICROPHONES
CAMERAS (X2)
LATERAL MICROPHONES (X2)
SHOULDER JOINT
SONARS (X2)
ELBOW JOINT
BATTERY
WRIST JOINT
TACTILE SENSORS
KNEE JOINT
SENSOR PRESSURE

AGRADECIMIENTOS:

Agradecer en primer lugar la ayuda prestada por mis tutores, Vicente Javier Julián Inglada y Carlos Carrascosa Casamayor, a las aclaraciones a mis numerosas preguntas, por su apoyo y por su comprensión.

Por supuesto, gracias a mis padres, por el aliento y el incondicional cariño y apoyo que siempre me han brindado en todas mis decisiones, y los valores que han sabido inculcarme para hacer de mi una mujer de provecho. A mis dos hermanas y a mi sobrina, porque siempre están ahí y saben cómo darme fuerzas en los momentos de flojeza.

Sin olvidarme de la madre de mi marido, que no es mi suegra, es mi segunda madre y además también es mi amiga, y así me lo demuestra día a día.

Y por último, pero no por ello menos importante, gracias a mi marido por su apoyo constante, su comprensión y por su paciencia.

A todos vosotros, ¡ Gracias !