UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

<Grado en ingeniería informática en Computadores>

Trabajo Fin de Grado

<Sava Drow>

<Nombre y Apellidos del Autor>

<Convocatoria, Año>

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

< Grado en ingeniería informática en Computadores >

Trabajo Fin de Grado

< Sava Drow >

Autor: <David Omar Flaity Pardo>

Tutor: <Antonio Silva Luengo>

Co-Tutor/es: <---->

**Tribunal Calificador**

Presidente: <Nombre y Apellidos>

Secretario: <Nombre y Apellidos>

Vocal: <Nombre y Apellidos>

# ÍNDICE GENERAL DE CONTENIDOS

[ÍNDICE GENERAL DE CONTENIDOS 3](#_Toc12834296)

[ÍNDICE DE TABLAS 6](#_Toc12834297)

[ÍNDICE DE ECUACIONES 8](#_Toc12834298)

[ÍNDICE DE FIGURAS 9](#_Toc12834299)

[RESUMEN 11](#_Toc12834300)

[SUMMARY 12](#_Toc12834301)

[INTRODUCCIÓN 13](#_Toc12834302)

[INICIOS 13](#_Toc12834303)

[OBJETIVOS 13](#_Toc12834304)

[INCÓGNITA INICIAL 14](#_Toc12834305)

[PLANTEAMIENTO PRELIMINAR 15](#_Toc12834306)

[PROCESO DE INVESTIGACIÓN: ESTADO DEL ARTE, IAS 16](#_Toc12834307)

[PRIMER CONTACTO 16](#_Toc12834308)

[REDES NEURONALES, EVOLUCIÓN E INCISO RAPIDO 17](#_Toc12834309)

[TIPOS DE CAPAS 18](#_Toc12834310)

[TIPOS DE REDES 19](#_Toc12834311)

[FUNCIONES DE ACTIVACION 20](#_Toc12834312)

[Aprendizaje por refuerzo, IAs generalistas 21](#_Toc12834313)

[Q-Learning 21](#_Toc12834314)

[Redes profundas Q y mejora sobre Q Learning 22](#_Toc12834315)

[TOP OF THE LINE HEURISTICAS: STOCKFISH 23](#_Toc12834316)

[Breakthrouging all the way: DEEPMIND 24](#_Toc12834317)

[ALPHAGO: ACERCAMIENTO UTILIZADO 25](#_Toc12834318)

[ENTRENAMIENTO 26](#_Toc12834319)

[ESTRUCTURA 27](#_Toc12834320)

[IMPLEMENTACIÓN Y METODOLOGÍA 29](#_Toc12834321)

[ITERACIÓN 1 – APP base 30](#_Toc12834322)

[ITERACIÓN 2 – Creación de sistema de comunicación entre clientes. 34](#_Toc12834323)

[ITERACIÓN 3 – Investigación y creación de IA. 38](#_Toc12834324)

[DATOS FINALES DEL PROCESO ITERATIVO 42](#_Toc12834325)

[DESARROLLO/IMPLEMENTACIÓN 45](#_Toc12834326)

[LIBRERÍA MULTIMEDIA 45](#_Toc12834327)

[CLASES PRINCIPALES 45](#_Toc12834328)

[LIBRERÍAS EXTERNAS 47](#_Toc12834329)

[APLICACIÓN: SAVA DROW 48](#_Toc12834330)

[Idea, definición y reglas 48](#_Toc12834331)

[Origen e historia del juego 51](#_Toc12834332)

[ESQUEMA DE LA APLICACIÓN 53](#_Toc12834333)

[ESTADÍSTICAS 62](#_Toc12834334)

[ESTADÍSTICAS DE ARCHIVOS LOCALES 62](#_Toc12834335)

[ESTADÍSTICAS DE CÓDIGO 63](#_Toc12834336)

[Estructuras de datos 67](#_Toc12834337)

[ALGORITMIA 69](#_Toc12834338)

[PROCESADO INICIAL, LUTS 69](#_Toc12834339)

[CAMINOS 71](#_Toc12834340)

[CONTROL DE IMÁGENES 72](#_Toc12834341)

[THREAD POOLING 73](#_Toc12834342)

[Gráficas y datos finales 74](#_Toc12834343)

[Sistema de servidor y cliente 79](#_Toc12834344)

[ESQUEMA DEL MODELO USADO 79](#_Toc12834345)

[Tabla de servidores públicos (API) 81](#_Toc12834346)

[TABLA DE MENSAJES 83](#_Toc12834347)

[FLUJO EN UNA GENERACIÓN/CONEXIÓN 84](#_Toc12834348)

[Inteligencia Artificial 85](#_Toc12834349)

[Algoritmo de fitness 85](#_Toc12834350)

[HEURÍSTICAS 86](#_Toc12834351)

[Decisores de movimientos 90](#_Toc12834352)

[Mejoras aplicadas 90](#_Toc12834353)

[Resultados algoritmos 91](#_Toc12834354)

[POSIBLES AMPLIACIONES 91](#_Toc12834355)

[Tablas transposicionales 91](#_Toc12834356)

[Redes neuronales, aprendizaje con refuerzo… 91](#_Toc12834357)

[COSTES 92](#_Toc12834358)

[CONCLUSIONES 93](#_Toc12834359)

[REFERENCIAS BIBLIOGRÁFICAS 94](#_Toc12834360)

# ÍNDICE DE TABLAS

[Tabla 1 – Hoja de ruta del proceso de búsqueda y adquisición de información 16](#_Toc12834464)

[Tabla 2 – Evolución de las versiones de Stockfish a lo largo del tiempo 23](#_Toc12834465)

[Tabla 3 – División de horas de trabajo por iteración y por etapa. 42](#_Toc12834466)

[Tabla 4 – Número de horas individuales dedicadas a cada etapa de cada iteración 43](#_Toc12834467)

[Tabla 5 – Número de horas acumulativas dedicadas 43](#_Toc12834468)

[Tabla 6 – Porcentajes de cada etapa sobre el total de cada iteración 44](#_Toc12834469)

[Tabla 7 - Porcentajes de las fases sobre el total 44](#_Toc12834470)

[Tabla 8 – Clases de **pygame** usadas en el ámbito gráfico 45](#_Toc12834471)

[Tabla 9 – Clases de **pygame** que controlan las interacciones del usuario y el resto del entorno (no gráfico) 46](#_Toc12834472)

[Tabla 10 – Clases menores de **pygame** utilizadas 46](#_Toc12834473)

[Tabla 11 – Librerías externas auxiliares 47](#_Toc12834474)

[Tabla 12 – Cifras de uso de librerías externas en la aplicación 47](#_Toc12834475)

[Tabla 13 – Tipos de personajes existentes en Sava Drow 49](#_Toc12834476)

[Tabla 14 – Elementos interactivos en Sava Drow 50](#_Toc12834477)

[Tabla 15 – Clases clave del módulo gráfico 55](#_Toc12834478)

[Tabla 16 – Clases clave del módulo de red 56](#_Toc12834479)

[Tabla 17 - Clases clave del módulo de IA 57](#_Toc12834480)

[Tabla 18 - Clases clave del módulo de Interacción (Elementos del tablero) 58](#_Toc12834481)

[Tabla 19 - Clases clave del módulo de Interacción (General) 59](#_Toc12834482)

[Tabla 20 - Clases clave del módulo de estructuras 60](#_Toc12834483)

[Tabla 21 - Clases clave del módulo de utilidades 61](#_Toc12834484)

[Tabla 22 – Tabla de tamaño conjunto de ficheros por tipo, ordenados por importancia 62](#_Toc12834485)

[Tabla 23 – Tabla de ficheros .py, ordenados por líneas de código 64](#_Toc12834486)

[Tabla 24 – Tabla de los módulos conformantes de la aplicación y sus estadísticas 64](#_Toc12834487)

[Tabla 25 – Tabla de ficheros .py, ordenados por módulo 66](#_Toc12834488)

[Tabla 26 - Gráfico de porcentajes de cada módulo 66](#_Toc12834489)

[Tabla 27 – Estructuras de datos utilizadas y sus propósitos 67](#_Toc12834490)

[Ilustración 28 – Ejemplo de cuadrantes en un tablero 68](#_Toc12834491)

[Tabla 29 – Pseudocódigo de distancias euclídeas 69](#_Toc12834492)

[Tabla 30 - Comparación de rendimiento usando un LUT para distancias euclídeas (Con tamaño 300\*300) 69](#_Toc12834493)

[Tabla 31 - Código usado para sacar el tiempo gastado en un acceso y en el cálculo de una raíz 69](#_Toc12834494)

[Tabla 32 - Pseudocódigo de cálculo de puntuaciones de movimiento en el tablero 70](#_Toc12834495)

[Tabla 33 - Comparación de rendimiento usando un LUT para almacenar las puntuaciones en cada turno 70](#_Toc12834496)

[Tabla 34 - Lut inicial para los caminos de un tablero 71](#_Toc12834497)

[Tabla 35 - Comparación de rendimiento usando un LUT para almacenar los caminos posibles 71](#_Toc12834498)

[Tabla 36 – Pseudocódigo de la estructura LUT de imágenes cargadas 72](#_Toc12834499)

[Tabla 37 – Ejemplo de uso de memoria según la técnica en uso 72](#_Toc12834500)

[Tabla 38 – Comparación de gasto de memoria y tiempo para todas las resoluciones 74](#_Toc12834501)

[Tabla 39 – Gasto de memoria con el tamaño de las superficies deslimitado (Resolución 1920\*1080) 75](#_Toc12834502)

[Tabla 40 - Ahorro de memoria y tiempo para todas las resoluciones, usando LUTs 75](#_Toc12834503)

[Tabla 41 - Comparativa de equipos de prueba 76](#_Toc12834504)

[Tabla 42 – Pruebas de rendimiento en los distintos entornos 77](#_Toc12834505)

[Tabla 43 – Comparación de rendimiento de las IAs (Media) (En resolución 640x360) 77](#_Toc12834506)

[Tabla 44 – Gráfica de comparación entre equipos (FPS y tiempos de carga) 78](#_Toc12834507)

[Tabla 45 - Gráfica de comparación entre equipos (Rendimiento heurísticas) 78](#_Toc12834508)

[Tabla 46 – Leyenda de los esquemas de red 79](#_Toc12834509)

[Tabla 47 – Ejemplo de dirección DNS de máquina virtual de Amazon. 82](#_Toc12834510)

[Tabla 48 – Mensajes usados en la conexión y sincronización de usuarios 83](#_Toc12834511)

[Tabla 49 – Mensajes utilizados en el transcurso de una sesión de juego 83](#_Toc12834512)

[Tabla 50 – Proceso de generación de una partida en un entorno de red (Clientes y Servidor) 84](#_Toc12834513)

[Tabla 51 – Pseudocódigo del algoritmo de poda alfa-beta utilizado 87](#_Toc12834514)

[Tabla 52 – Pseudocódigo del algoritmo de búsqueda MonteCarlo utilizado 89](#_Toc12834515)

[Tabla 53 – Tabla de generadores de movimientos en tablero no heurísticos 90](#_Toc12834516)

[Tabla 54 – Cambios realizados en el pseudocógido de alfa beta al incluir ordenación 90](#_Toc12834517)

[Tabla 55 – Comparación de decisores de movimiento 91](#_Toc12834518)

[Tabla 56 – Comparación de rendimiento de heurísticas según tiempo 91](#_Toc12834519)

[Tabla 57 – Costes finales del proyecto 92](#_Toc12834520)

# ÍNDICE DE ECUACIONES

[Ecuación 1 – Fórmula de Bellman 21](#_Toc11178720)

[Ecuación 2 – Fórmula del multiplicador de peligro 79](#_Toc11178721)

[Ecuación 3 - Fórmula de la puntuación de peligro 79](#_Toc11178722)

[Ecuación 4 – Fórmula de la puntuación de captura 79](#_Toc11178723)

[Ecuación 5 – Fórmula de la puntuación de cebo 79](#_Toc11178724)

[Ecuación 6 – Fórmula final del algoritmo de fitness 80](#_Toc11178725)

[Ecuación 7 – Fórmula del límite de confianza superior aplicado a árboles usado en MonteCarlo 82](#_Toc11178726)

# ÍNDICE DE FIGURAS

[Ilustración 1 - Captura de la aplicación 14](#_Toc12834420)

[Ilustración 2 - Logo de la aplicación en la pantalla inicial 15](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834421)

[Ilustración 3 – Esquema de neurona artificial 17](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834422)

[Ilustración 4 – Salida de una capa de agrupación 18](#_Toc12834423)

[Ilustración 5 – Diagrama de una convolución 18](#_Toc12834424)

[Ilustración 6 – Red neuronal con dos capas totalmente conectadas (1º y 3º) 18](#_Toc12834425)

[Ilustración 7 y 8 – Izquierda: RN Convolucional, Centro: RN Prealimentada, Derecha: RN recurrente 19](#_Toc12834426)

[Ilustración 9 – Representación gráfica de funciones de activación (Animado en Link fuente). 20](#_Toc12834427)

[Ilustración 10 – Grafo representativo del proceso de decisión de Markov 21](#_Toc12834428)

[Ilustración 11 – Izquierda: Esquema básico de una Red profunda-Q, Derecha: Arquitectura más optimizada 22](#_Toc12834429)

[Ilustración 12 – Comparación de rendimiento en un entorno con 100 acciones posibles. 22](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834430)

[Ilustración 13 – Ramificación en una partida de GO. Cada estado del juego tiene 150-250 movimientos posibles. 25](#_Toc12834431)

[Ilustración 14 – Primera etapa del entrenamiento de AlphaGo 26](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834432)

[Ilustración 15 – 2ª etapa del entrenamiento de **AlphaGo** 26](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834433)

[Ilustración 16 – 3ª etapa del entrenamiento de AlphaGo 26](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834434)

[Ilustración 17 – Estado del juego, AlphaGo 27](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834435)

[Ilustración 18 – Estructura de la red neuronal de **AlphaGo** 27](#_Toc12834436)

[Ilustración 19 – Cabecera de ajuste de valores, AlphaGo 28](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834437)

[Ilustración 20 – Cabecera de política a seguir, AlphaGo 28](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834438)

[Ilustración 21 – Iteración de un proceso iterativo e incremental 29](#_Toc12834439)

[Ilustración 22 – Esquema completo del proceso iterativo 29](#_Toc12834440)

[Ilustración 23 – Comparación entre esquema previo y resultado final 48](#_Toc12834441)

[Ilustración 24 – Esquema simplificado de la estructura de la aplicación 53](#_Toc12834442)

[Ilustración 25 – Diagrama UML del módulo gráfico (Excepto UIElement) 54](#_Toc12834443)

[Ilustración 26 - Diagrama UML del módulo gráfico (Solo UIElement) 54](#_Toc12834444)

[Ilustración 27 – Diagrama UML del módulo de red (Python) 56](#_Toc12834445)

[Ilustración 28 – Diagrama UML del módulo de red (JavaScript) 56](#_Toc12834446)

[Ilustración 29 - Diagrama UML del módulo de IA 57](#_Toc12834447)

[Ilustración 30 - Diagrama UML del módulo de interacción 58](#_Toc12834448)

[Ilustración 31 - Diagrama UML del módulo de estructuras 60](#_Toc12834449)

[Ilustración 32 - Diagrama UML del módulo de utilidades 61](#_Toc12834450)

[Ilustración 33 – Ejemplo de puntuaciones ‘fitness’ en el tablero 70](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834451)

[Ilustración 34 – Atributos de un objeto Restricción de un personaje 71](#_Toc12834452)

[Ilustración 35 - Atributos de un objeto ResizedSurface de una imagen 72](file:///C:\Users\Hinjeniero\sava_drow\documentation\tfe_epcc_tfg_plantilla_memoria.docx#_Toc12834453)

[Ilustración 36 - Flujo de paso de las tareas mediante Thread Pooling 73](#_Toc12834454)

[Ilustración 37 – Representación gráfica del proceso (Agrupamiento de hilos) 73](#_Toc12834455)

[Ilustración 38 – Esquema de comunicación en una conexión a un servidor privado 80](#_Toc12834456)

[Ilustración 39 – Esquema de comunicación en una conexión a un servidor privado 80](#_Toc12834457)

[Ilustración 40 – Todos los puntos de acceso del servidor de Express 81](#_Toc12834458)

[Ilustración 41 – Panel de ajustes de **Noip** 82](#_Toc12834459)

[Ilustración 42 – Panel de control de instancias en el servicio web de Amazon (AWS) 82](#_Toc12834460)

[Ilustración 43 – Ejemplo de ejecución MiniMax 86](#_Toc12834461)

[Ilustración 44 – Representación gráfica del árbol resultante tras una ejecución de poda alfa beta 87](#_Toc12834462)

[Ilustración 45 – Representación gráfica proceso del MCTS 89](#_Toc12834463)

# RESUMEN

Los objetivos de este **TFG** se resumen en los siguientes puntos.

* Investigación del estado del arte en **agentes no humanos de control del comportamiento**, ya sean basados en redes neuronales o en heurísticas, así como la creación de un agente de este tipo.
* Creación de un sistema de comunicación a través de la red, que permita la **interacción entre dos o más clientes**, que ejecutan una misma aplicación.
* Estudio de **algoritmia compleja**, comprendiendo búsqueda de caminos, creación de elementos gráficos compuestos, y control de imágenes con estructuras de tipo **LUT (look up table)**.

Para satisfacer estos objetivos, se ha realizado una aplicación gráfica propia, para poder construir los aspectos prácticos necesarios, y alcanzar los objetivos anteriormente mencionados.

La **metodología** de desarrollo se ha basado en el proceso iterativo e incremental, creando distintas versiones de la aplicación, y añadiendo funcionalidades a la vez que se cumplía los objetivos en cada iteración.

El trabajo se ha enfocado, en la parte práctica, en tener un sistema y una estructura funcionales en una aplicación que fuera, al menos, interesante y distinto a todo lo que hay públicamente. De esta manera, este trabajo podría servir para llenar el portafolio de trabajos propios del graduado, así como un proyecto público que pudiese ser usado, con un fin ocioso.

En la porción de indagación, se prioriza la fracción sobre inteligencia artificial, al ser de las ciencias más novedosas, útiles, y de moda, que existen en la actualidad. La exploración ha sido extensa, y para la implementación de los últimos sistemas, se requiere una cantidad de tiempo y materiales que sobrepasan la situación actual del autor de este trabajo. Sin embargo, se han implementado sistemas jugadores con suficiente complejidad para que sea interesante crear y analizar.

En conclusión, como autor, he adquirido conocimientos sobre las ramas que me interesan, y creado un programa satisfactorio en el camino.

# SUMMARY

The initial objectives of my final graduation project (This one that you are reading), include the following ones:

* Investigation over the current state of the art and cutting edge advancements in **non human behavior** **control agents**, whether they are based on neural networks with deep learning, or heuristic algorithms.
* The building of a communication system through the network, that allows interaction **between two or more users**, executing the same application.
* Study and implementation of complex algorithms, that comprehend paths exploration and search, multiple image control and management, a complex builder of graphic elements, and the use of **LUT tables (look up table).**

To meet those goals, I have succeeded in creating my own graphic application, that provides the essential structure to support the creation of the necessary systems.

Talking about development methodology, I have followed the iterative and incremental process. This one follows a spiral, returning a working application version in each loop, and adding features and functionality over this one in the next. We reached the targets in the way, or course.

The main focus of this project has been, in the practical part, **to create and sustain a working application** that is interesting, and different from everything that is out there (publicly, at least), and of my own making. Like this, it also serves the purpose of filling this graduate’s portfolio, and the purpose of being a fun application that people will use of their own volition.

Regarding researching, the artificial intelligence fraction has been given priority, due to its recent peak in popularity, usefulness and future prospects. After the process, we can conclude that the implementation of the latest kind of systems and breakthroughs, have requirements (of time and resources) that the alumni cannot possible satisfy in his currently situation. In spite of that, working non human subsystems, with enough complexity to be worth to review it, have been added.

In conclusion, as the author, I have acquired knowledge over the branches that interest me, and created a satisfactory program along the journey.

# INTRODUCCIÓN

## INICIOS

Desde hace décadas, dos de las ramas de la informática más desarrolladas, han sido la comunicación entre dispositivos, y la automatización de acciones o simulación de comportamiento humano.

En la actualidad los avances son evidentes:

Cualquier desarrollador tiene la posibilidad de establecer un sistema de intercambio de información para su aplicación, por un precio mínimo o incluso nulo, sin necesidad de que los dispositivos a conectar estén en el mismo lugar físico. Recientemente, incluso se puede incluir un servidor en la red local o en un lugar remoto a un coste asequible a cualquiera.

El estado del arte de la inteligencia artificial ha cambiado y evolucionado a pasos agigantados, sobre todo en los últimos 5 años. Paradigmas que no tenían solución a la vista, se han encontrado con un plan establecido o incluso con una solución ya desarrollada y disponible al público.

Estos sectores tienen una presencia muy extendida e imprescindible actualmente en el ámbito de la informática. Por ello, hemos decidido que este Trabajo de Fin de Grado (TFG) sea el desarrollo de una aplicación que implemente sistemas pertenecientes a cada sección comentada antes.

## OBJETIVOS

Este capítulo tiene la función de presentar los problemas y objetivos del proyecto, profundizando en los retos que nos pueda plantear cada uno de ellos.

En el planteamiento se divisan los siguientes objetivos:

**1**.- **Desarrollo e implementación de una aplicación** o programa, que nos proporcione un entorno sobre el que construir los sistemas posteriores.

La idea sobre la que se realice el desarrollo deberá tener un componente de novedad, para hacerlo más interesante, tanto para el proceso de programación como para el resultado final.

**2**.- **Creación y funcionamiento de un sistema de comunicación** entre dos o más usuarios a través de la red, mediante mensajes internos JSON, que consigue que todos los extremos sean conscientes de la interacción que mantienen, ya que se muestran las distintas acciones externas en la aplicación.

**3**.- **Investigación del estado del arte actual en la materia de inteligencia artificial**, comprendiendo esto estructuras de datos como capas neuronales de distintos tipos, que forman redes neuronales complejas, algoritmos heurísticos con guardado de tablas que transponden entre cada ejecución de los mismos, y una mezcla de ambos para conseguir el mayor resultado.

Al ser una ciencia de rápido avance y cambio, se hacen indispensables un proceso de aprendizaje previo, y un estudio de los últimos avances en la materia antes de lanzarse a la carga con la producción.

**4**.- La **creación y funcionamiento de agentes no humanos** en un entorno propio, cuyo comportamiento lógico está basado en algoritmos de automatización de distinta naturaleza, con mejoras para incrementar el resultado conseguido por dichos algoritmos.

En rasgos generales, la idea es tener un producto final funcional, que contenga los dos sistemas creados y funcionando sobre él: **Comunicación** entre usuarios, y **automatización** mediante agentes no humanos.

## INCÓGNITA INICIAL

Inicialmente, el primer problema que te encuentras es que hacer de aplicación base, así como las limitaciones que debe tener el entorno resultado, ya que después debe soportar todas las funcionalidades ya discutidas.



Ilustración - Captura de la aplicación

## PLANTEAMIENTO PRELIMINAR

Para ello, se decide hacer una **aplicación** gráfica que contenga un juego de mesa basado en un juego llamado **Sava**, que aparece en las novelas de **LoveCraft**.

Este juego tiene aspectos que son parecidos al clásico ajedrez, como son el sistema por turnos y casillas, y el pertenecer a la categoría de juegos con información perfecta (Aquellos en los que ambos contrincantes conocen en todo momento el estado del tablero al completo).

Ilustración - Logo de la aplicación en la pantalla inicial

Teniendo aspectos que se diferencian lo suficiente como para no ser una repetición de todo lo que ya hay desarrollado e investigado, pero con el parecido suficiente para permitir el uso de lógica y datos resultantes de proyectos de ajedrez.

**Python** como lenguaje y **Pygame** como librería gráfica, son los elegidos para el desarrollo.

Hablando de **comunicación**, el protocolo **JSON** es el elegido para el formato de mensajes e información, al ser una de las estructuraciones más extendidas y utilizadas en el envío y recepción a través de la red.

Los **Sockets** son el mecanismo base de conexión entre dispositivos, ya implementados en la librería utilizada como soporte, **MasterMind**.

Por último, el proceso de investigación y aprendizaje de **inteligencia artificial** consta de varias etapas, siendo la primera el estudio y aprendizaje de las estructuras y algoritmos que se utilizan, así como sus funciones objetivo y casos de uso.

Seguimos con la indagación en el estado del arte actual, y la hoja de ruta finaliza con la decisión e implementación de un sistema de decisiones automatizado en la aplicación, que siga la lógica correspondiente.

Este trabajo tiene tanto un componente de implementación y desarrollo, como otro de investigación y estudio del estado del arte.

El resultado tendrá la finalidad de cumplir el máximo porcentaje posible de los objetivos establecidos, dentro de las restricciones de tiempo, conocimiento y costes que nos imponga la situación actual.

# PROCESO DE INVESTIGACIÓN: ESTADO DEL ARTE, IAS

## PRIMER CONTACTO

Al buscar información sobre inteligencia artificial y sus últimas tendencias, lo primero que se encuentra son redes neuronales, Aprendizaje Profundo (**Deep Learning**), Aprendizaje por refuerzo (**Q Learning**) inteligencia artificial generalista… y la empresa de origen británico, y posteriormente adquirida por Google, **DeepMind**.

Pero para que, al leer posts, artículos, mirar esquemas…puedas entenderlos, falta mucho camino todavía, y bastante base. Primero hay que averiguar la información base sobre las redes neuronales desde sus orígenes y su evolución.

Los siguientes apartados contienen el grueso de información que se ha descubierto durante este proceso, dividida en definiciones y clasificaciones de las distintas estructuras, y el funcionamiento de los últimos avances aplicables a nuestro producto.

La hoja de ruta del aprendizaje e investigación se ha ordenado según conceptos, siguiendo el siguiente esquema.

Tabla – Hoja de ruta del proceso de búsqueda y adquisición de información

## REDES NEURONALES, EVOLUCIÓN E INCISO RAPIDO

Hagamos primero un pequeño y rápido inciso (Ya que no atañe directamente a este documento), en la historia, definición y evolución de las **Redes Neuronales.**

Las denominadas Redes Neuronales tuvieron su origen como idea en 1943. Son un sistema computacional que se basa parcialmente en las neuronas humanas, cuyo primer modelo práctico fue el **Perceptron**, creado en 1958.

Si simplificamos, una red neuronal no es más que una serie de matrices, que se multiplican entre ellas, dando una matriz resultado.

|  |
| --- |
| CAPA DE ENTRADA  CAPA OCULTA  W11  W21  CAPA DE SALIDA  W12  O1  W21  W11  X1  W31  W22  W13  X2  W23  Ilustración 2 – Perceptron Multicapa |

Cada neurona que forma la red devuelve un valor, que puede ser el resultado de un cálculo (En capas intermedias y finales), o la entrada directamente (Capas de entrada). A su vez, individualmente, cada conexión recibida tiene un peso asociado (otro valor numérico). Interviene también el **Bias** (Compensación), que no es más que otra cifra recibida.

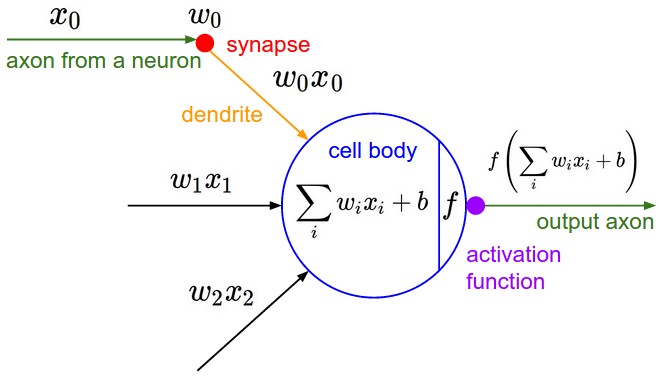
[](http://cs231n.github.io/assets/nn1/neuron_model.jpeg)Por último, cada unidad de la estructura basará su valor de disparo o activación (**triggering)**, en una función de activación decidida de antemano, y el valor de entrada en esa unidad.

Ilustración – Esquema de neurona artificial

## TIPOS DE CAPAS

Capas de agrupación (**Pooling**), que permiten que la red neuronal reconozca patrones ya vistos, aunque la entrada esté en una posición distinta al ejemplo de entrenamiento.

[](http://cs231n.github.io/assets/cnn/pool.jpeg)[](http://cs231n.github.io/assets/cnn/maxpool.jpeg)

Ilustración – Salida de una capa de agrupación

Capas de convolución (**Convolutional**), actúan de filtro de una entrada para extraer las características interesantes de dicha entrada, teniendo normalmente una salida menos extensa que la entrada.

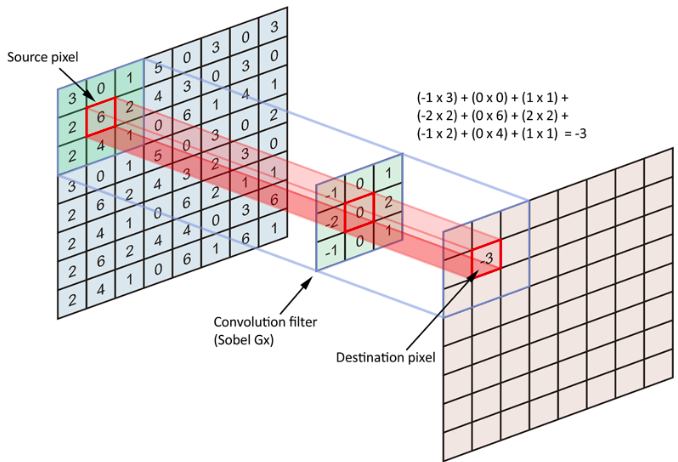
[](https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwi_p5CphcniAhUExYUKHdwPAzoQjRx6BAgBEAU&url=https%3A%2F%2Fwww.freecodecamp.org%2Fnews%2Fan-intuitive-guide-to-convolutional-neural-networks-260c2de0a050%2F&psig=AOvVaw00NeuiujYyqDq5ZMG14oSK&ust=1559504386431045)

Ilustración – Diagrama de una convolución

Capa totalmente conectada (**Fully-Connected**), consigue que a red neuronal aprenda combinaciones no lineales, imprescindible en problemas con un mínimo de complejidad.

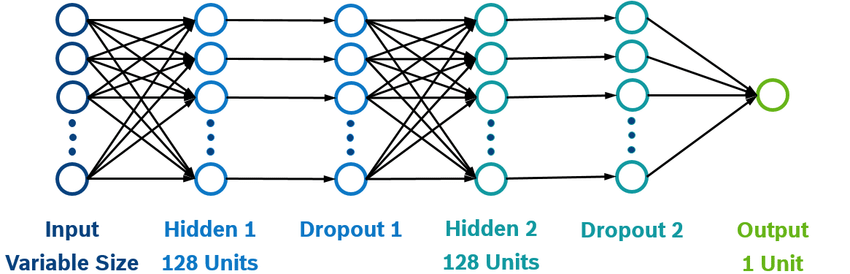
[](https://www.researchgate.net/profile/Igor_Gilitschenski/publication/311920717/figure/fig2/AS:617625239425024@1524264740078/Sample-dense-neural-network-with-2-fully-connected-layers-2-dropout-layers-and-a.png)

Ilustración – Red neuronal con dos capas totalmente conectadas (1º y 3º)

## TIPOS DE REDES

Se han investigado más tipos de redes, pero por el bien de la brevedad, se describen sólo las más usadas (RN == Red Neuronal):

RN prealimentada **(FFN)**: Funcionan *dirigiendo las salidas* de una capa *hacia las entradas* de la siguiente, sin muchas más complicaciones.

Las redes de este tipo son las más enfocadas a ser de propósito general, ya que pueden dar con la solución correcta siempre que cuenten con suficientes neuronas, capas, ejemplos y entrenamiento. Aunque son propensas a quedarse en mínimos locales.

RN Convolucional **(CNN)**: Su finalidad principal es la extracción de características clave de diversas entradas, ya que si los datos de entrada superan cierto tamaño, es inviable su procesado en una RN sin reducirlos.

En las conexiones iniciales entre capas, se realiza un mapeo no-lineal, que consigue identificar los patrones más “fuertes”, que son reforzados o no tras pasar por todo el set de entrenamiento.

Están formadas por capas de convolución, de agrupación, y algunas neuronas de **Perceptron** para la clasificación final.

RN recurrente **(RNN)**: Para reconocimiento de patrones, como es reconocimiento de imágenes o lenguaje natural. Cogen una única función, y la aplican a una secuencia de datos de entrada.

En cada iteración se actualiza su comprensión de dichos datos. Esto es perfecto para el análisis de series de datos, y se usan incluso para la generación de los mismos (Predicciones, por ejemplo), aunque sufren de problemas como pérdida de memoria acerca de ejemplos antiguos.

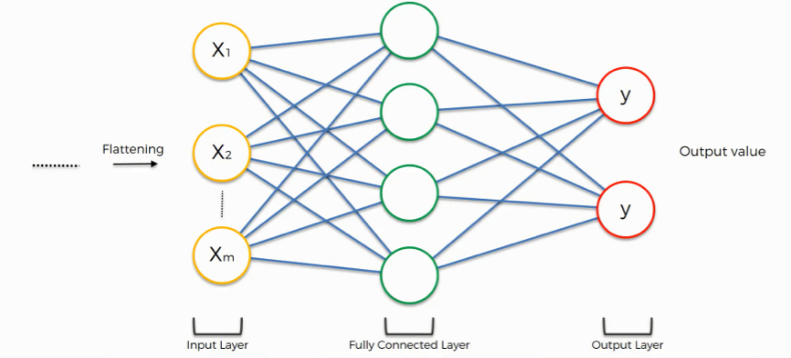
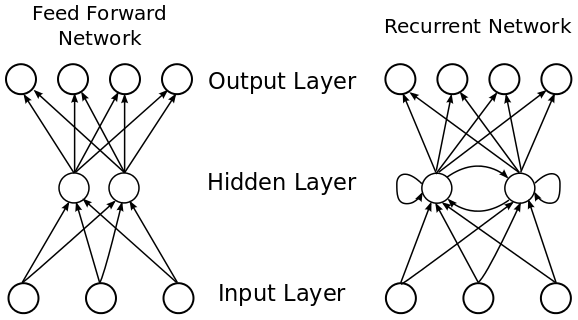
[](https://sds-platform-private.s3-us-east-2.amazonaws.com/uploads/74_blog_image_1.png)[](https://www.researchgate.net/profile/Dana_Hughes3/publication/305881131/figure/fig5/AS:391681317851147@1470395511494/Feed-forward-and-recurrent-neural-networks.png)

Ilustración y – Izquierda: RN Convolucional, Centro: RN Prealimentada, Derecha: RN recurrente

## FUNCIONES DE ACTIVACION

Las funciones de activación tienen la utilidad de

Aquellas usadas de forma generalista, han ido cambiando según la época y sus necesidades. En un inicio se usaban la binaria, y la identidad o función linear .

Al pasar a problemas y paradigmas más complejos, surge la necesidad de usar funciones no lineales, por lo que se empiezan a ver la Sigmoide y la Tangente Hiperbólica.

Pero ninguna está libre de problemas.   
La **sigmoide** provoca que los gradientes entre capas desaparezcan, al saturarse con facilidad (Muy cerca de 0 o de 1). Tampoco tiene su centro en 0, lo que provoca que, al devolver siempre un resultado positivo, lo que provoca que la optimización sea más complicada, ya que, en la etapa de “compensación”, los gradientes evolucionan en direcciones distintas.

La **Tangente Hiperbólica** tiene más utilidad práctica, al tener sus valores entre -1 y 1, pero también sufre del problema de desvanecimiento de gradientes.

Por ello, recientemente se usa como función de activación en prácticamente la totalidad de problemas (En las capas ocultas), la Unidad Lineal Rectificada con Fuga (o **leaky ReLU** en inglés), que mejora la ya potente Unidad Lineal Rectificada **(ReLU**) (Posee una convergencia hasta X6 más rápida que la tangencial).

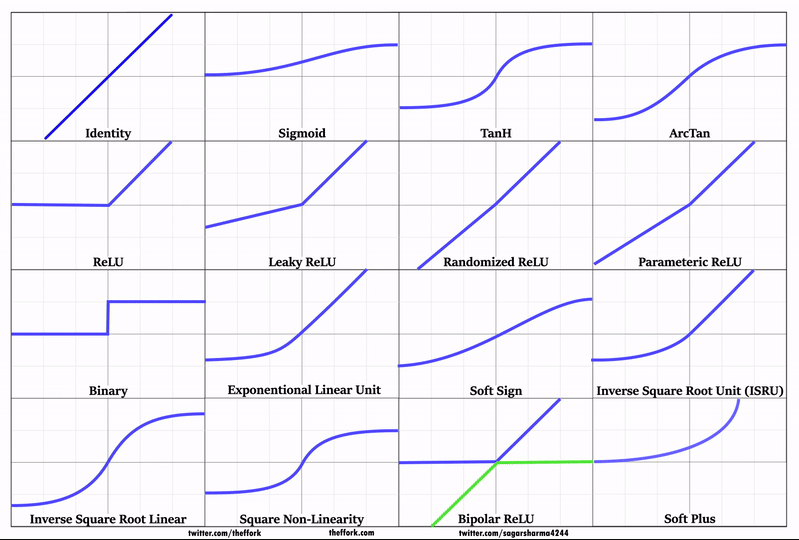
[](http://theffork.com/wp-content/uploads/2019/03/1.gif)

Ilustración – Representación gráfica de funciones de activación (Animado en Link fuente).

Aprendizaje por refuerzo, IAs generalistas

Este tipo de aprendizaje sigue un proceso, que determina que acciones escoge un agente en el entorno proporcionado, con el fin de maximizar la “recompensa” producto de dicha decisión (Mayoritariamente elegida mediante el **proceso de decisión de Markov**).

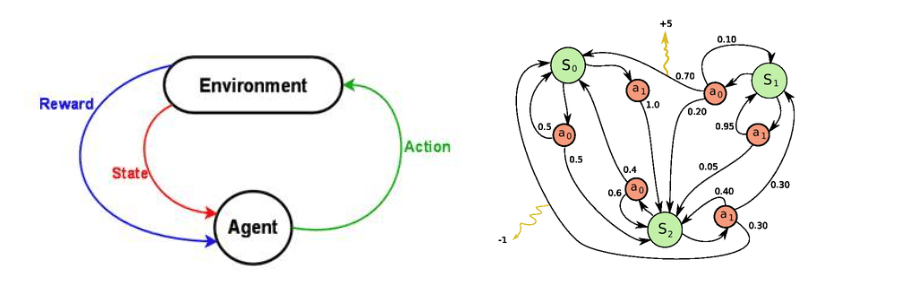
[](https://www.intel.ai/demystifying-deep-reinforcement-learning/)

Ilustración – Grafo representativo del proceso de decisión de Markov

### Q-Learning

Nos centramos en la técnica de Aprendizaje de calidad, o **Q-Learning** (Q de Quality). Básicamente es el procesado de la relación acción-recompensa, y su posterior guardado y evolución.

En este aprendizaje, debemos definir una función Q(s, a), que devuelve “La máxima recompensa futura ajustada, cuando realizamos la acción a en el estado s”. A esta función se le llama función de calidad (Q-function).

Inicialmente, se crea la **tabla de calidad** en el arranque (**Q-Table)**, que no es más que una matriz que contiene un valor Calidad[estado, acción] en cada posición, e inicializamos todas las posiciones a cero. Después, el proceso avanza mediante la interacción sucesiva del agente con el entorno, lo cual produce actualizaciones en la **tabla de calidad**.

En la elección de acciones manejamos dos tipos de posibilidades: **Exploración** y **explotación**. Normalmente usamos un equilibrio entre las dos, utilizando aleatoriedad (para explorar el espacio de acciones), y la propia tabla con los datos recopilados como referencia (explotando lo ya adquirido).

A partir de aquí, hay varianza en la propia actualización de la tabla en cada iteración, lo cual nos hará converger el agente hacia el comportamiento deseado.

Para calcular la máxima recompensa para una decisión y estado, se usa la ecuación de ***Bellman***, que nos permite aproximar las **funciones de calidad** de la tabla hacia la convergencia.

Ecuación – Fórmula de Bellman

### Redes profundas Q y mejora sobre Q Learning

Debido al elevado número de estados que puede contener un entorno, en todas las situaciones reales, se utiliza una **red neuronal** que sustituye la tabla de funciones de calidad, que sería inimaginablemente grande en la situación descrita.

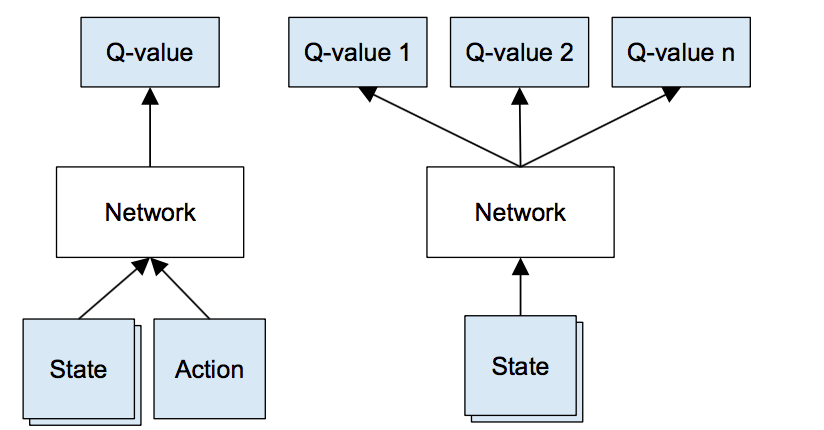
[](http://simplecore-dev.intel.com/ai/wp-content/uploads/sites/71/deep-q-network-example.png)

Ilustración – Izquierda: Esquema básico de una Red profunda-Q, Derecha: Arquitectura más optimizada

Las aplicaciones reales que usan *Q-L* contienen mejoras, ya que tienden a sobrestimar el potencial de las acciones en un estado concreto. Y en este caso, esta acción será más propensa a ser elegida en iteraciones siguientes, dificultando que el agente tenga una exploración uniforme para encontrar la convergencia adecuada.

Una solución es tener **dos redes neuronales**, separando efectivamente la selección de la acción de todo el procesado y actualización de los valores Q de la tabla. A esto se le llama Aprendizaje de calidad doble (**Double Q-Learning**).

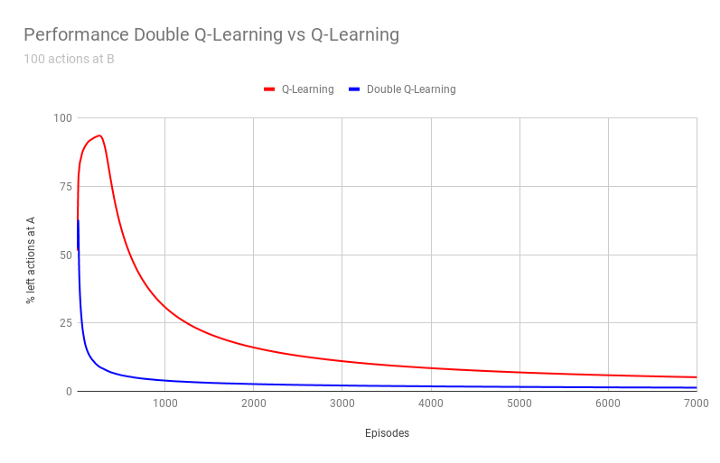
[](https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3)

Ilustración – Comparación de rendimiento en un entorno con 100 acciones posibles.

## TOP OF THE LINE HEURISTICAS: STOCKFISH

Pasemos a hablar de heurísticas. Al buscar los procedimientos y las bases usadas en agentes no humanos, en entornos parecidos al de nuestra aplicación, surgen los algoritmos de **tipo heurístico** (Ya conocidos debido a asignaturas anteriores en la propia carrera).

Salen a la luz al indagar sobre una de las más famosas y exitosas inteligencias artificiales de ajedrez: StockFish**.**

Desde el lanzamiento de su primera versión estable en *2008*, este motor de ajedrez ha evolucionado sin descanso hasta su última versión estable v10, a fecha de **mayo de 2019**.

Historia al margen, parte de las técnicas usadas consisten en heurística, concretamente en Podado Alfa Beta(El más conocido), así como un conjunto de métodos de la misma categoría para movimientos de apertura y demás, incluyendo también mejoras sobre el propio alfa beta, como tablas de transposición.

En resumen, este descubrimiento abre los ojos a las posibilidades de los métodos heurísticos en mi trabajo, que, con diversos mejoras y estructuras, obtienen un rendimiento que no esperaba que rivalizase con las técnicas más modernas dependiendo del entorno.

Tabla 2 – Evolución de las versiones de Stockfish a lo largo del tiempo

<http://ccrl.chessdom.com/ccrl/404/rating_list_all.html>

<https://www.chessprogramming.org/Stockfish>

## Breakthrouging all the way: DEEPMIND

Ya con más conocimientos, llegamos al grueso de la cuestión, y son logros y novedades prácticas en el campo de la inteligencia artificial. Centramos el foco en la empresa DeepMind: TODO This could be a graphic hierarchy

En 12/2013, presentan una IA que juega a 6 programas distintos de la Atari 2600 sin cambiar la estructura o el algoritmo de aprendizaje. Es una IA GENERALISTA. Usa los pixeles de la pantalla como entrada, y como técnica, aprendizaje por refuerzo. A fecha 02/2015, aplican este modelo a 49 juegos distintos, y consiguen un desempeño superhumano en todos ellos.

<https://arxiv.org/abs/1312.5602>

<https://www.intel.ai/demystifying-deep-reinforcement-learning/#gs.ex6y01>

10/2015, el motor AlphaGo gana a un campeón europeo de GO. Una IA pensando a un nivel profesional en GO era considerado como uno de los problemas sin resolver, ya que es un juego con un árbol de decisiones gigantesco, incluyendo hasta 300 posibilidades desde cada estado de tablero.

En 05/2017, **AlphaGo** vence al que fue campeón mundial de GO durante 2 años. El entrenamiento de este algoritmo consiste en un protocolo de aprendizaje supervisado, usando gran cantidad de datos extraídos de partidas humanas.

Se produce una versión mejorada, llamada AlphaGo Zero, que tiene como peculiaridad su aprendizaje, siendo éste totalmente autónomo y sin supervisión, simplemente programando las reglas del juego. El periodo de tiempo de entrenamiento fue también mucho más corto (Sólo 3 días versus varios meses en el caso previo).

El modelo nuevo fue enfrentado al antiguo en 2017, arrojando unos resultados de 100 victorias a 0, a favor de la novedad.

A finales de 2017 sale a la luz AlphaZero, una modificación de ***AlphaGo Zero*** pero adaptada para poder controlar cualquier juego de 2 jugadores con información perfecta. Nos centraremos en el funcionamiento de este algoritmo.

07/2018, investigadores de ***DeepMind*** entrenan uno de sus sistemas para jugar al modo “Captura la bandera” de Quake 3 (Juego en 3 dimensiones),

<https://arxiv.org/abs/1807.01281>

<https://www.youtube.com/watch?v=MvFABFWPBrw>

08/2018, presentan una IA que denominan ‘generalista’ llamada Impala. Aunque no sustituye a un humano, es un paso intermedio importante. Una inteligencia artificial generalista es básicamente el cerebro humano, que es el mayor hito a alcanzar en este campo.

<https://www.youtube.com/watch?v=u4hf4uZnZlI>

La última hazaña tiene de nombre AlphaStar, y tiene como objetivo StarCraft 2, un juego con aspectos de estrategia en tiempo real. Con un entrenamiento mixto, inicialmente basado en repeticiones de humanos, y posteriormente con partidas contra sí mismo, ya tiene el nivel de un jugador profesional.

<https://en.wikipedia.org/wiki/DeepMind>

## ALPHAGO: ACERCAMIENTO UTILIZADO

Al ser una evolución de **AlphaGo Zero**, la estructura utilizada es similar, por lo que usamos el esquema liberado públicamente, para entender el sistema usado y la organización interna del algoritmo.

La imagen original ha sido traducida por el autor de este documento.

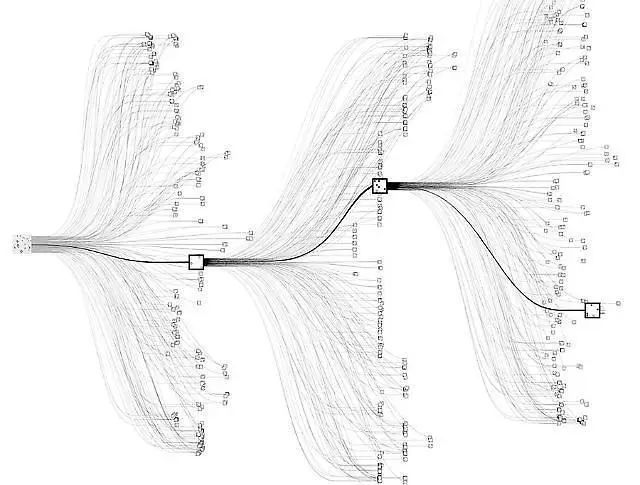
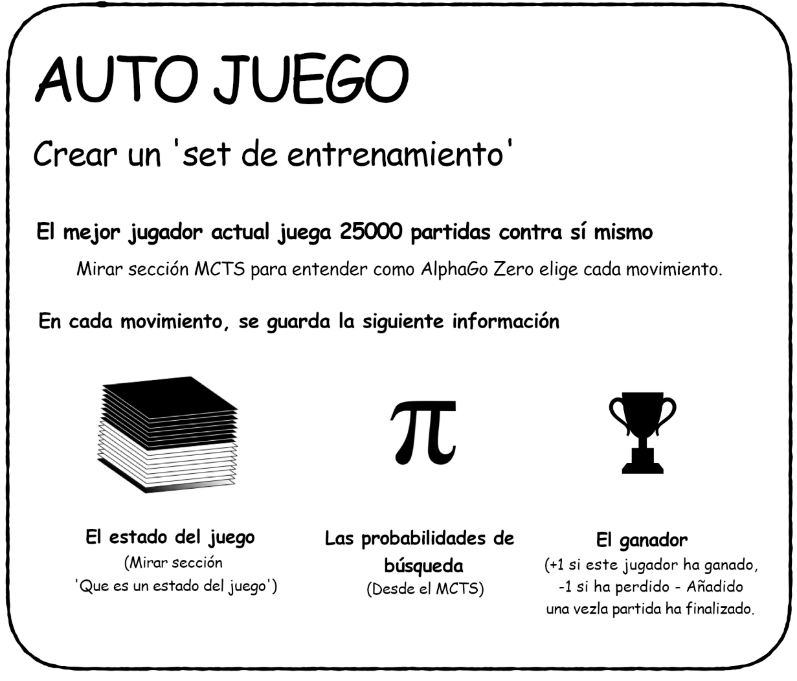


Ilustración – Ramificación en una partida de GO. Cada estado del juego tiene 150-250 movimientos posibles.

### ENTRENAMIENTO

En primer lugar tenemos el proceso de entrenamiento, que consiste en un bus de 3 etapas, ejecutadas en paralelo:

El fundamento principal del entrenamiento es la ausencia de presencia humana.

La red neuronal aprende ‘tabula rasa’, es decir, desde un estado en blanco, sin conocimientos o movimientos expertos.

Se crea el set para el ajuste de la red neuronal en base a partidas contra sí mismo.



Ilustración – Primera etapa del entrenamiento de AlphaGo

De los ejercicios anteriores, se coge   
una batería de posiciones para   
reentrenar la red y ajustarla.

Los valores de las posiciones analizadas  
son normalizados, para eliminar   
irregularidades.

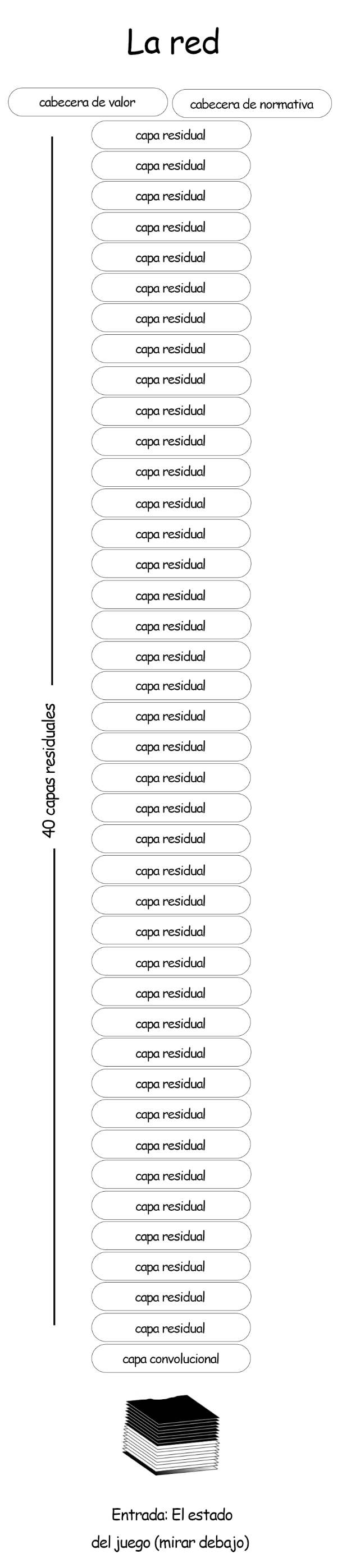
Ilustración – 2ª etapa del entrenamiento de **AlphaGo**

Se estima la nueva red resultante de los pasos anteriores enfrentándola a   
 la última red resultado.

Según la conclusión, se establece la  
 nueva red como nuevo mejor jugador  
 o no.

Ilustración – 3ª etapa del entrenamiento de AlphaGo

### ESTRUCTURA



…

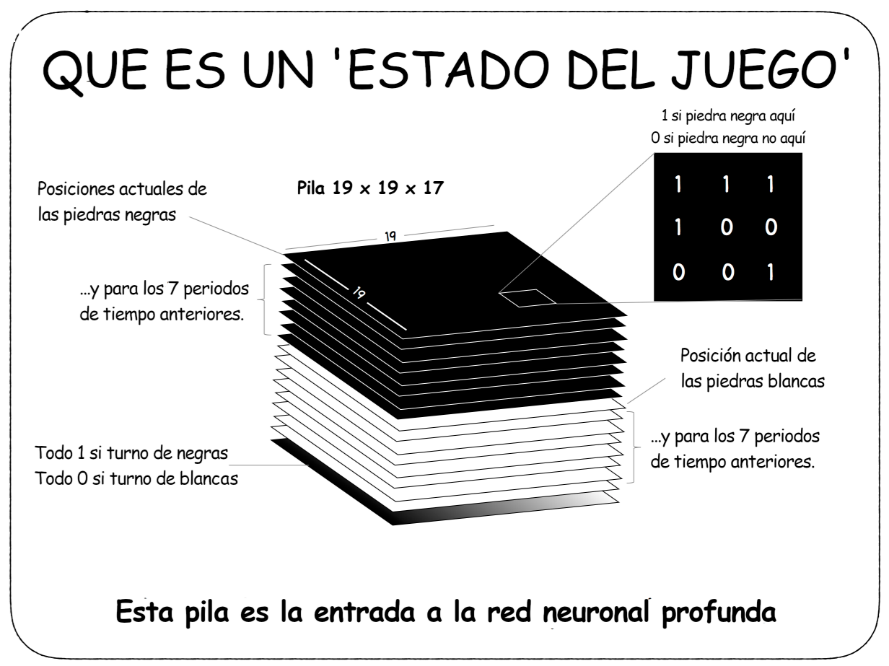
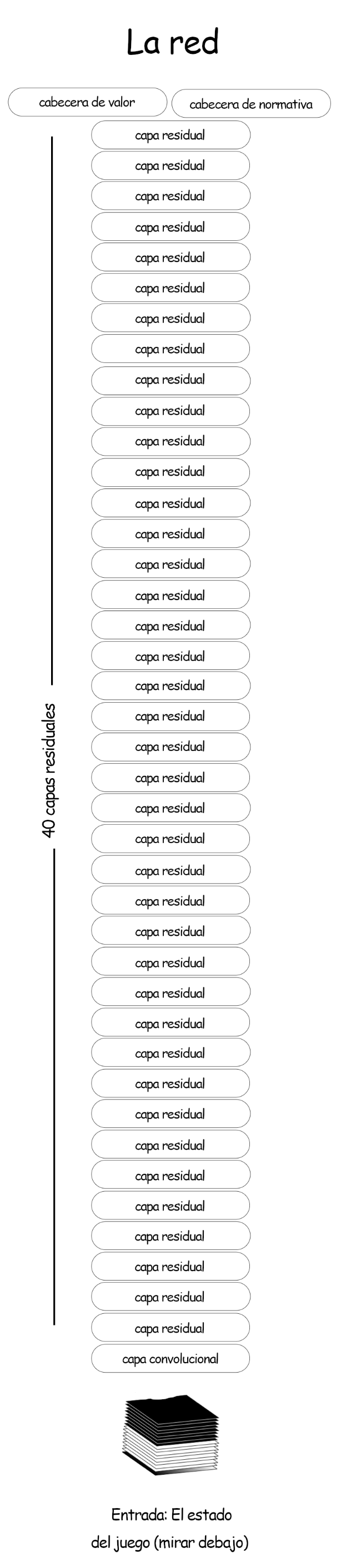


Ilustración – Estado del juego, AlphaGo



…

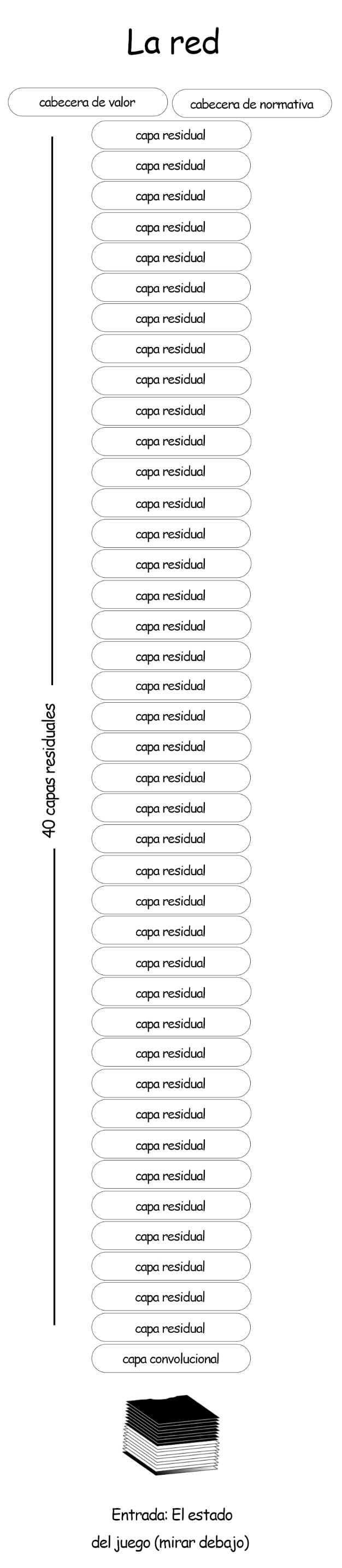


Ilustración – Estructura de la red neuronal de **AlphaGo**

En las ilustraciones se vé como se codifica un estado del juego para usarlo de entrada en la red neuronal, y todas las capas de la red en sí (De forma invertida).

Las capas convolucionales reducen el tamaño de los datos mediante filtros, y las 40 capas residuales ….

Las dos cabeceras finales normalizan y ajustan la salida, en función de la política que se esté siguiendo en ese momento.   
Sus contenidos se muestran gráficamente a continuación.

Servirán para cerrar el capítulo que explica los interiores de AlphaGo

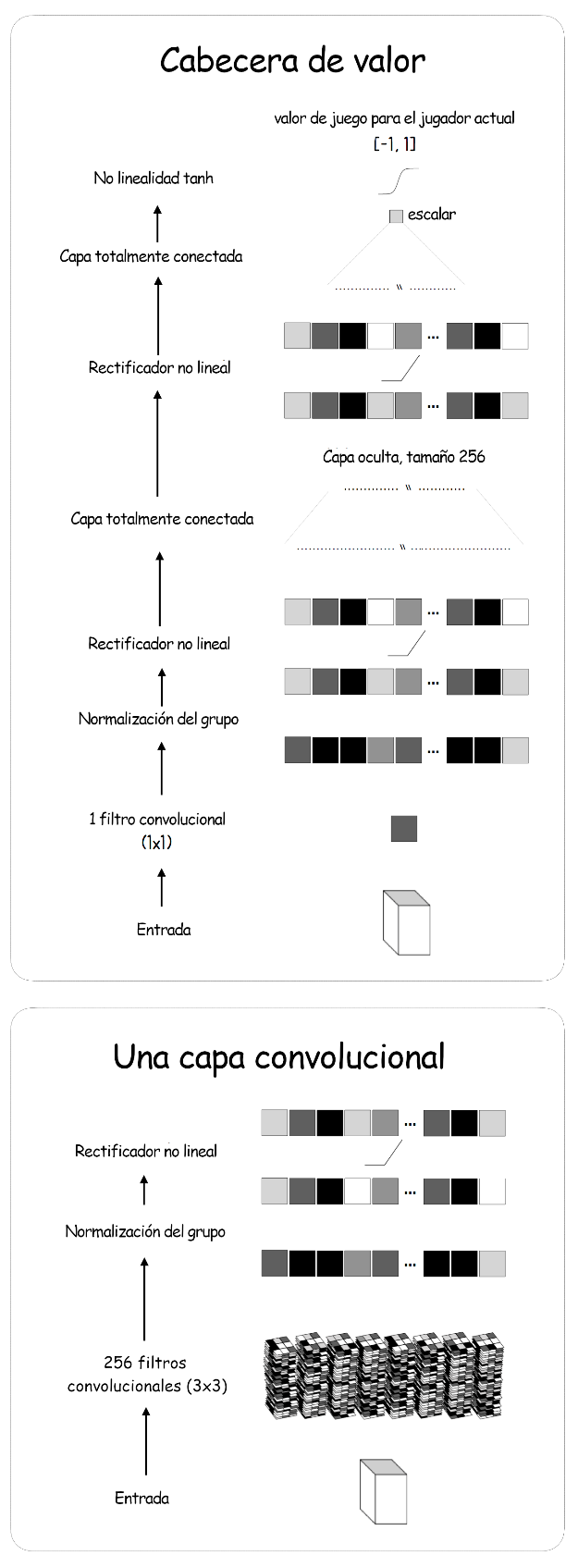
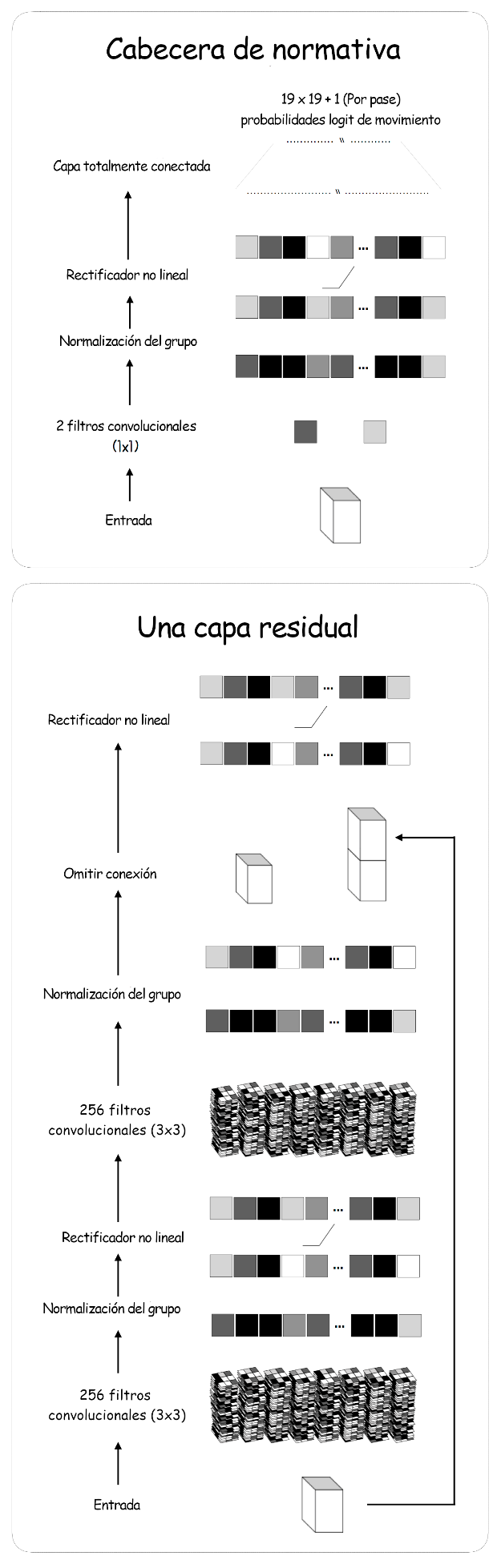


Ilustración – Cabecera de ajuste de valores, AlphaGo

Ilustración – Cabecera de política a seguir, AlphaGo

# IMPLEMENTACIÓN Y METODOLOGÍA

Se ha seguido la metodología ágil **Iterativa e Incremental.**

El proceso a seguir consiste en realizar los pasos básicos de desarrollo de software durante varias iteraciones, produciendo al final de cada una un resultado funcional.

La diferencia básica entre la metodología en **Espiral** y ésta, es la idea de iteración. En espiral, cada iteración está basada en un tiempo determinado. Aquí la iteración acaba al liberar el producto con los requisitos implementados (Correspondientes a esa etapa del bucle).

N = N+1

**ANÁLISIS**

**DISEÑO**

**IMPLEMENTACIÓN**

ITERACIÓN N

**PRUEBAS**

**INTEGRACIÓN**

Ilustración – Iteración de un proceso iterativo e incremental

En el caso de este producto, se divide el proceso al completo en 3 iteraciones importantes, cada una llevada a cabo en un plazo calculado en la tabla final.

Ilustración – Esquema completo del proceso iterativo

## ITERACIÓN 1 – APP base

Indagación en librerías que soporten la creación de aplicaciones multimedia

Análisis de objetivos

Elección y análisis librería

Análisis de requisitos

**ANÁLISIS**

1 hora

1 hora

5 horas

2 horas

En el análisis de objetivos y requisitos se estudian los fines del trabajo y los requisitos para poder alcanzarlos. Como requisitos entendemos una planificación que no excluye cosas como estructuras que necesitamos, lenguaje a usar…

La indagación de librerías con su posterior elección tuvo como factores de peso principales una curva de aprendizaje corta, y facilidad de uso.

Como no importa si la aplicación gráfica es en 2 dimensiones en lugar de 3, y saliendo su última versión poco antes de la época en la que tuvo lugar este paso, se eligió **pygame**. En su análisis se estudió la documentación de la librería.

**DISEÑO**

1 hora

Diseño del esqueleto de la aplicación

Diseño gráfico de la aplicación, incluyendo distintas ventanas

2 horas

Diseño de las estructuras de datos necesarias

1 hora

Diseño completo de objetos, jerarquías y relaciones

4 horas

Primer diseño, se realizó un esquema previo de la aplicación con los objetos sueltos.

El diseño gráfico se hizo en papel, describiendo la idea inicial del aspecto gráfico de las pantallas (Menú, juego…), y el flujo entre ellas.

Los dos últimos pasos comprenden la creación (en papel), de un esquema previo completo de la aplicación (Que en la etapa de implementación es modificado ligeramente según necesidades), incluyendo los métodos de cada objeto y todos los objetos necesarios, incluso los menos importantes.

**IMPLEMENTACIÓN**

Implementación de algoritmos importantes

Diseño de algoritmos importantes

Implementación del primer prototipo provisional (objetos provisionales)

Diseño de clases de manejo de elementos gráficos

Implementación de clases de manejo de elementos gráficos

Reestructuración de jerarquías y objetos, completado de clases, e interconexión de todos los elementos implementados.

30 horas

5 horas

40 horas

5 horas

50 horas

60 horas

El producto de esta etapa será visto más en profundidad en la sección de Desarrollo, aquí nos centraremos en los tramos en cuestión. La lista superior no está organizada de forma cronológica estricta, ya que se ha saltado entre algunos pasos.

El primer prototipo simplemente es el andamio de la aplicación, la creación de objetos del diseño completo realizado. No es funcional aún, tan sólo es la jerarquía de clases y métodos base.

Los algoritmos más complejos y necesarios, a los que pertenecen, por ejemplo, la búsqueda de caminos, el control de superficies, las tablas LUT… se muestran como un paso aparte por su elevada carga de trabajo por sí mismos.

**PRUEBAS**

**INTEGRACIÓN**

Test y pruebas de algoritmos de caminos y grafos

20 horas

Test y pruebas de clases de control y manejo de elementos gráficos

20 horas

10 horas

Test y pruebas de algoritmos de manejo de superficies y archivos

Test y pruebas de aplicación completa, e interacción entre sus distintos elementos

50 horas

Cada punto de control mostrado arriba, incluye en el propio concepto y en las horas, el proceso de **debug** de cada parte, así como todas las reparaciones y añadidos al código realizados.

Los tests consisten en la ejecución controlada del elemento en cuestión (Si es necesario, incluyendo alguna estructura auxiliar para mostrar más tarde), y el ensayo de todas las posibles situaciones, así como su fix.

## ITERACIÓN 2 – Creación de sistema de comunicación entre clientes.

**ANÁLISIS**

Indagación en librerías auxiliares de comunicación entre clientes

Análisis de objetivos de comunicación

2 horas

4 horas

Elección de librería y análisis

2 horas

Inicialmente, se observan los fines de interacción necesarios entre clientes que estén en distintos ordenadores, y en base a ellos, se redacta una lista de requisitos, como son latencia baja, comunicaciones estables, no mucha sobrecarga de la red…

A continuación, se buscan librerías que ayuden con el trabajo, teniendo los niveles más bajos de los procesos de envío y recepción cubiertos y testeados. Se buscan para Python y también en la página oficial de **pygame**.

Finalmente, se eligió la librería **MasterMind**, que se puede encontrar en la página oficial de **pygame**. Tras un breve análisis de la misma y su código, se llegó a la conclusión de que era nos daba suficiente base para nuestros propósitos.

<http://www.pygame.org/project-Mastermind+Networking+Lib-859-.html>

<https://geometrian.com/programming/projects/index.php?project=Mastermind%20Networking%20Library>

**DISEÑO**

Diseño del esquema de la capa de comunicación de la aplicación

2 horas

2 horas

Esquema de mensajes necesarios

Diagrama de flujo de mensajes y su intercambio

2 horas

Esquematizado de toda la capa de comunicación y su estructura (Se decide en un diagrama **N cliente 🡪 1 servidor**), de la clasificación de mensajes imprescindibles, y de su flujo entre clientes y el servidor.

El restante del diseño abarca el máximo posible de cuestiones que se pueden ver sin in en la codificación en sí.

**IMPLEMENTACIÓN**

10 horas

Programación del prototipo provisional (objetos servidor-cliente)

Creación y envío de mensajes (en JSON)

5 horas

Clasificación de mensajes en cliente y servidor, control de su flujo y sincronización de los partícipes

20 horas

Reestructuración de la aplicación, para la Integración de elementos online en el producto de la iteración anterior

15 horas

El prototipo inicial contiene los objetos de servidor y tablero cliente, con los métodos de conexión, pero sin información real que compartir.

Tras crear los distintos tipos de mensajes, así como su manejo en cliente/servidor (incluyendo sincronización entre participantes), se conecta todo lo creado con la aplicación, para poder usarse de forma práctica.

Test de creación de clientes y servidores, y conexión entre ambos

**PRUEBAS**

**INTEGRACIÓN**

4 horas

4 horas

2 horas

Pruebas de sincronización y espera entre clientes

Pruebas de envío y recepción de mensajes entre host y clientes

Ensayo completo de partidas multicliente en red local

20 horas

Ensayo completo de partidas multicliente en red WAN

30 horas

Para toda esta batería de pruebas hicieron falta dos equipos, ya que el objetivo de esta iteración era conseguir conectar más de un cliente con el resto, estando éstos físicamente separados.

Como curiosidad, si algún fallo provocada en cierre forzado del programa, usando **Windows** como SO, el puerto permanecía ocupado, por lo que para iniciar de nuevo el programa era necesario buscar en el administrador de procesos el hilo del servidor en cuestión y acabar con su ejecución manualmente.

El ensayo fuera de red local ha sido bastante más engorroso, al tener que conectar uno de los dispositivos de prueba a una red totalmente externa a donde tuviésemos el servidor y el otro cliente.

## ITERACIÓN 3 – Investigación y creación de IA.

**ANÁLISIS/INVESTIGACIÓN**

10 horas

Investigación del panorama actual en Inteligencia artificial

Estudio de las IAs más utilizadas en los casos de uso más parecidos al nuestro

20 horas

Adquisición de conocimientos sobre las IAs con más posibilidades de ser útiles para nuestro proyecto

20 horas

Segunda indagación de los últimos avances en IA con vistas a implementación Inteligencia artificial

20 horas

Elección de técnicas a usar y tipo de IA a implementar, en case a predicción de trabajo y tiempo

5 horas

Los resultados de esta fase son evidentes en el capítulo anterior de ‘**Estado del Arte, Inteligencia Artificial’**.

Los puntos de control del proceso son autoexplicativos, y no requieren ser analizados.

**DISEÑO**

3 horas

Esquematizado de algoritmos a usar

Estudio y diseño de los algoritmos heurísticos

10 horas

Diagrama de flujo entre algoritmos y el resto del programa

2 horas

10 horas

Diseño de los dos algoritmos de puntuación de movimientos

En este diseño solo es reseñable hablar del estudio y diseño de los algoritmos, que consiste en el aprendizaje del comportamiento de los dos algoritmos heurísticos elegidos (**Alfa-beta** y **Monte Carlo**), y su posterior pseudocódigo aproximado, realizado en papel.

Podemos separar el diseño de los métodos que asignan una puntuación a cada movimiento proporcionado en la entrada.

En este método se apoyan de forma casi completa las ias básicas y de forma auxiliar las jodidas

**IMPLEMENTACIÓN**

Implementación del algoritmo auxiliar de puntuación de movimientos (fitness)

20 horas

Implementación de los algoritmos heurísticos que formarán la base de los agentes no humanos

20 horas

Creación de métodos auxiliares necesarios para manejas las estructuras de los algoritmos anteriores

15 horas

Programación de IAs básicas, e Integración con el entorno resultante de la última iteración

12 horas

Creación de métodos heurísticos y de los métodos básicos de generación de movimientos, de los métodos auxiliares para manejar las simulaciones en ambas IAs complejas, y de ambos algoritmos de puntuación de movimientos.

Para finalizar, se integra todo lo programado en la aplicación, y se añaden los elementos necesarios para poder cambiar de ajustes cuando sea deseado.

**PRUEBAS**

**INTEGRACIÓN**

Pruebas del funcionamiento completo de los algoritmos de fitness

10 horas

5 horas

Ensayo completo de partidas con jugadores controlados por ordenador en tablero LAN/WAN

Ensayo completo de partidas con jugadores controlados por ordenador en partida local

Test de algoritmos básicos de participantes no humanos

30 horas

20 horas

30 horas

5 horas

Pruebas de las heurísticas y modificaciones de las mismas debido a errores

Ensayo de la integración de los ajustes de esta iteración con el resto de la APP

Fase final de pruebas usando los diferentes tipos de generadores de movimientos para entes no humanos.

## DATOS FINALES DEL PROCESO ITERATIVO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ITERACIÓN | ETAPA | HORAS | HORAS ACUMULADAS EN ITERACIÓN | HORAS ACUMULADAS TOTALES |
| 1 (APP) | **Análisis** | **9** | **9** | **9** |
| 1 (APP) | **Diseño** | **8** | **17** | **17** |
| 1 (APP) | **Implementación** | **190** | **207** | **207** |
| 1 (APP) | **Pruebas** | **100** | **307** | **307** |
| 2 (Red) | **Análisis** | **8** | **8** | **315** |
| 2 (Red) | **Diseño** | **6** | **14** | **321** |
| 2 (Red) | **Implementación** | **50** | **64** | **371** |
| 2 (Red) | **Pruebas** | **60** | **124** | **431** |
| 3 (CPU) | **Análisis** | **75** | **75** | **506** |
| 3 (CPU) | **Diseño** | **25** | **100** | **531** |
| 3 (CPU) | **Implementación** | **67** | **167** | **598** |
| 3 (CPU) | **Pruebas** | **100** | **267** | **698** |
|  | | | | |

Tabla – División de horas de trabajo por iteración y por etapa.

Tabla – Número de horas individuales dedicadas a cada etapa de cada iteración

Tabla – Número de horas acumulativas dedicadas

Tabla – Porcentajes de cada etapa sobre el total de cada iteración

Exceptuando el análisis de la iteración de inteligencias artificiales (Ya que incluye también investigación y estudio, procesos laboriosos), la mayor parte del tiempo de una ejecución completa (sumando hasta un 90%), se invierte en las etapas de implementación y pruebas, de media, más en la primera.

Tabla - Porcentajes de las fases sobre el total

# DESARROLLO/IMPLEMENTACIÓN

## **LIBRERÍA** **MULTIMEDIA**

[Pygame](https://www.pygame.org/docs/tut/newbieguide.html), la librería de elección para nuestra aplicación, es una capa (**wrapper**) en **Python** para [SDL](http://libsdl.org/) (**Simple DirectMedia Layer**), una librería multiplataforma de desarrollo, que proporciona acceso de bajo nivel a elementos como sonido, teclado, ratón, y procesadores de gráficos mediante **OpenGL** y **Direct3D**.

Con este acceso se nos posibilita la creación de un programa multimedia interactivo, a la vez que se simplifican las comunicaciones con los periféricos y elementos de proceso.

Esto nos permite programar mediante **Python**, cualquier aplicación que pudiese ser implementada usando **SDL**, en cualquiera de sus plataformas soportadas (en nuestro caso **Windows**).

### CLASES PRINCIPALES

Algunas de las clases más usadas en la aplicación.

Tabla – Clases de **pygame** usadas en el ámbito gráfico

Tabla – Clases de **pygame** que controlan las interacciones del usuario y el resto del entorno (no gráfico)

Tabla – Clases menores de **pygame** utilizadas

### LIBRERÍAS EXTERNAS

Tabla – Librerías externas auxiliares

|  |  |  |  |
| --- | --- | --- | --- |
| Librería externa | Clases contenedoras | Llamadas en código | Llamadas en ejecución(Est) |
| *Gradients* | 1 | 3 | ~30 |
| *PText* | 1 | 2 | ~100 |
| *Bezier* | 1 | 0 | 0 |
| *PADLIB* | 1 | 1 | ~10 |
| *Mastermind* | 2 | ~45 | ~500 |

Tabla – Cifras de uso de librerías externas en la aplicación

## APLICACIÓN: SAVA DROW

### Idea, definición y reglas

El objetivo final de una sesión del juego es capturar a la Matrona de todos los contrincantes, mediante el uso de cualquiera de las fichas que queden libres.

El entorno está formado por un tablero con forma de red de araña, con un número establecido de casillas en las que se da la acción entre personajes, ya sea simplemente el traslado de los que controlas o la captura de enemigos.

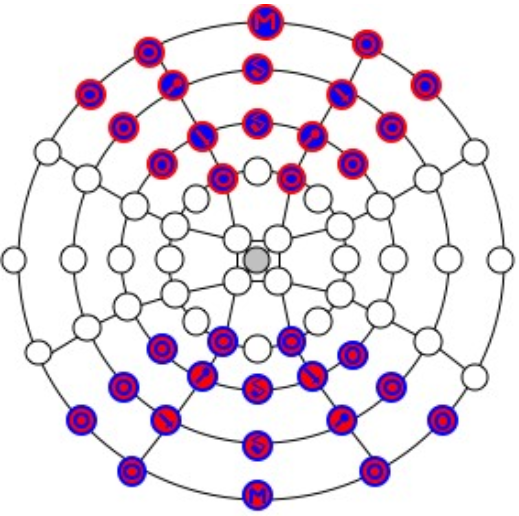
[](https://docs.google.com/document/pub?id=1BgkLwn66qN0oej3QEnjjcF-4LunZpjKTA62uQlETTBw)



Ilustración – Comparación entre esquema previo y resultado final

El marco de ejecución comprende multitud de elementos interactivos, los cuáles son explicados a continuación, tanto su función como sus acciones posibles.

#### PERSONAJES

Son las distintas piezas que cada jugador tiene en su poder, siendo el elemento con mayor protagonismo, y que se traslada a lo largo del tablero.

Cada personaje tiene un rango de movimientos posibles y un valor establecido inicialmente, que irá aumentando según la cantidad de personajes de ese tipo concreto disminuya. **La representación puede variar.**

Tabla – Tipos de personajes existentes en Sava Drow

#### Elementos del tablero

El resto de elementos aquí mostrados cumplen con una función cualquiera, ya sea albergar a los personajes, mostrar información del jugador o del estado actual de la aplicación, o activar alguna función.

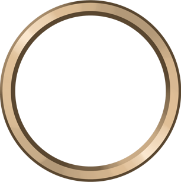


Tabla – Elementos interactivos en Sava Drow

La casilla puede tener 3 diseños distintos, dependiendo de los ajustes, y el Dado también aparece al principio para sortear el orden de los turnos.

Como apunte adicional, el intercambio o “**upgrade**” del Orco en las casillas de promoción, solo ocurre si hay personajes de mayor valor capturados. En caso contrario no ocurre nada.

### Origen e historia del juego

Sava es un juego creado por los Drow. Comprende el movimiento y captura de piezas (que representan soldados) en un estilo de juego que recuerda al ajedrez.

Este es el juego que Lolth and Eilistraee son vistos jugando en la trilogía “Lady Penitent”, con el tablero representando un plano de la existencia al completo.

Los orígenes de Sava son misteriosos. De acuerdo a una leyenda popular, fue un regalo de la diosa *Lolth* a la raza de los *Ilythiiri*, como vía para enseñarles los tejidos de la sociedad de los *Drow*: Traición, suerte, astucia y estrategia.

Sin embargo, la mayoría de los historiadores calculan su aparición poco después de “The scattering”, alrededor de -3000DR, ya que no se ha encontrado evidencia de su existencia en periodos previos.

A medida que los *Drow* se acomodaban a un estilo de vida más pacífico, urbano y menos expansionista, el aumento del tiempo libre de la nobleza provocó el desarrollo de *Sava* como pasatiempo.

Como era, y es aún, útil para enseñar muchas lecciones útiles sobre la supervivencia como *Drow*, las matronas de muchos clanes dieron implícitamente su visto bueno, e incluso algunas incluso alentaban a sus subordinados. Estaba considerado una parte vital de la educación de cualquier líder de pelotón, y durante un tiempo se jugaba como un rito religioso en la iglesia de *Lolth*, en *Sshamath*.

En los siglos recientes, la popularidad de sava entre los nobles ha decrecido, en parte debido al incremento de jugadores de clases inferiores.   
Ya que los únicos requisites para tener habilidad en *sava* son una mente ágil y conocimiento de las normas, varios *Drow* de clase humilde fueron capaces de dominar el juego y vencer a miembros de las clases nobles.

Esta situación fue considerada intolerable por el Archimago *Gromph Baenre* quien, tras ser derrotado en un duelo contra un simple soldado raso, con calma declaró:

“…el perder contra tí no me hace el *Drow* inferior. Pero, sin embargo, me enfurece muchísimo” – Tras lo cual convirtió al desafortunado ganador en un champiñón.

Como resultado de su universalidad, *sava* se ha vuelto menos común y popular en los salones de los nobles, reducción más que compensada con su diseminación en el resto de la sociedad *Drow*.

Incluso existen jugadores *no-Drow* – El mago *Elminster* es conocido por ser un contrincante competente, aunque pocos (si no ninguno) de éstos se ha enfrentado a un maestro de los *Drow*.

Ésto se debe parcialmente a la reclusividad general de la raza *Drow*, pero sobre todo sucede porque los elfos oscuros protegen con ferocidad su juego tradicional, y tienden a despellejar y/o eviscerar miembros de otras razas si los encuentran participando en una partida.

En cualquier caso, *sava* puede decirse que está completamente arraigado en la cultura de los *Drow*, y que ningún *no-Drow* podría nunca desear entender completamente sus aspectos más sutiles.

*Sava* es un eje fundamental en el mundo de los elfos oscuros. Aunque no lo admitirían jamás, el hecho de que esté disponible tanto para pobres como ricos, lo convierte en un factor unificador en una sociedad que, de otra forma, estaría fuertemente estratificada.

Se ha dicho que el mundo de los *Drow* es uno de Caos, traición y desorden – irónicamente, el juego que incluye estos conceptos bien podría ser parte de la unión que mantiene dicho mundo junto.

**Sava** tiene su origen en la inferencia de la partida que Lolth y Eilistraee juegan en la trilogía ‘Lady Penitent’, con el tablero representando todo un plano de la existencia.

Dicha trilogía forma parte a su vez de ‘The Forgotten Realms’, un mundo de ficción alternativo de Dragones y Mazmorras.

<https://forgottenrealms.fandom.com/wiki/Sava>

<https://docs.google.com/document/pub?id=1BgkLwn66qN0oej3QEnjjcF-4LunZpjKTA62uQlETTBw>

## ESQUEMA DE LA APLICACIÓN

Debido al tamaño y la estructura del programa, se mostrará primero la interacción entre grandes apartados (módulos), y más tarde se dividirán esos módulos en las clases componentes de los mismos.

Ilustración – Esquema simplificado de la estructura de la aplicación

Debido a la complejidad de las dependencias entre las distintas clases y métodos, los módulos de la ilustración superior no están divididos de forma perfecta, ya que algunas clases se encargan de cambios gráficos y de las interacciones del usuario, por ejemplo. Pero como esquema inicial es suficiente.

Realizamos un tour por los distintos módulos, mostrando diagramas UML y explicando cada clase en profundidad.

En algunos de los diagramas existen clases que se repiten. Esto es debido a que sus propósitos están mezclados en varios módulos, pero solo se entrará en ellas en más profundidad en el módulo del que tenga más carga dicha clase.

Red e Inteligencia Artificial serán explicados más en detalle en sus respectivas secciones.

#### Módulo gráfico

Ilustración – Diagrama UML del módulo gráfico (Excepto UIElement)

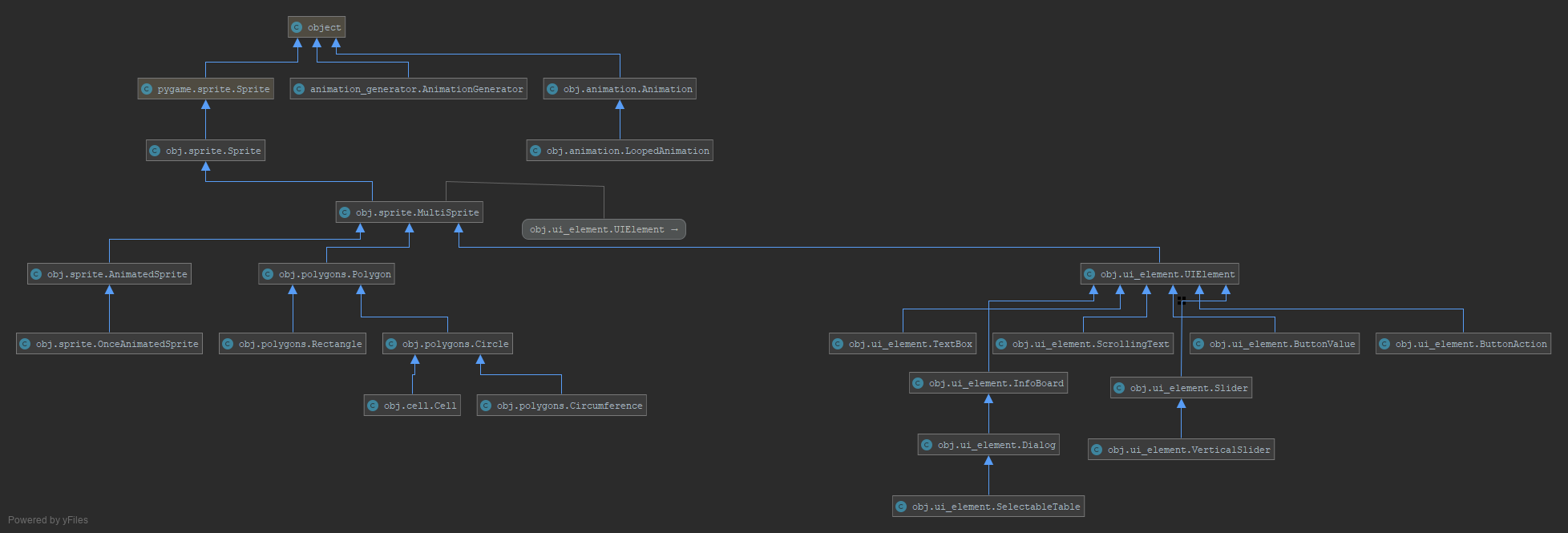


Ilustración - Diagrama UML del módulo gráfico (Solo UIElement)

Tabla – Clases clave del módulo gráfico

#### Módulo de red

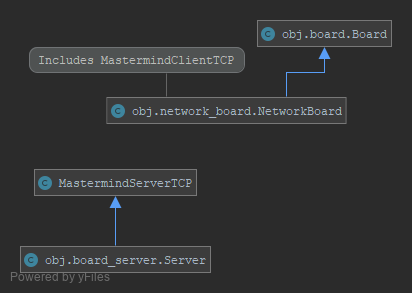


Ilustración – Diagrama UML del módulo de red (Python)

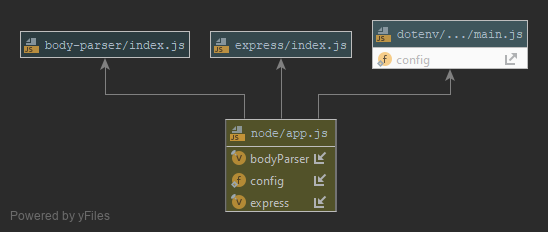


Ilustración – Diagrama UML del módulo de red (JavaScript)

Tabla – Clases clave del módulo de red

#### Módulo de inteligencia artificial

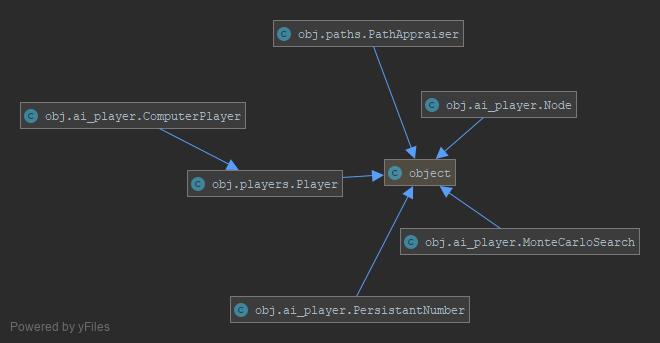


Ilustración - Diagrama UML del módulo de IA

Tabla - Clases clave del módulo de IA

#### Módulo de interacción (lógica)

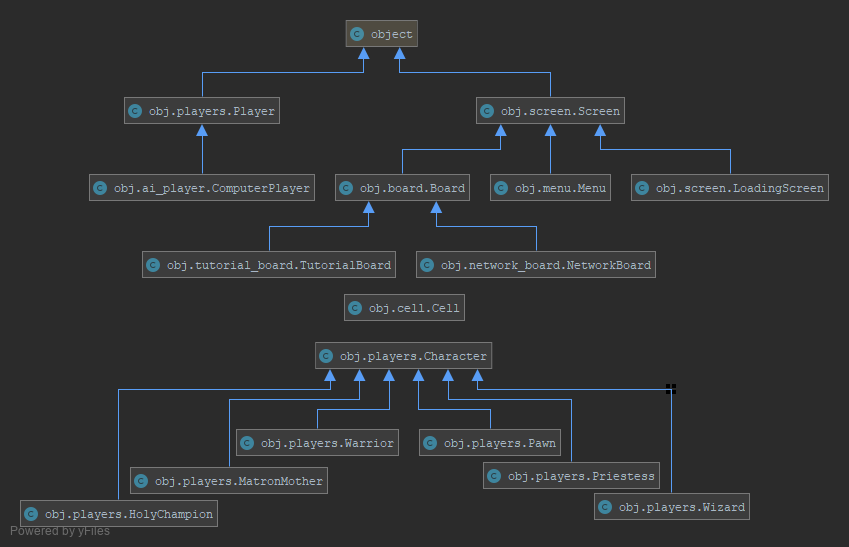


Ilustración - Diagrama UML del módulo de interacción

Tabla - Clases clave del módulo de Interacción (Elementos del tablero)

Como se comprueba en el esquema y la explicación de las clases, en el módulo de interacción no se encuentran clases que tan sólo manejan eventos de entrada del usuario, si no que controlan la lógica y actúan en consonancia. La gran mayoría de ellas también tiene cierta carga gráfica.

Por ejemplo, en **Board**, se crean varios elementos como las casillas y las circunferencias, y se muestran en pantalla.

Tabla - Clases clave del módulo de Interacción (General)

#### Módulo de estructuras

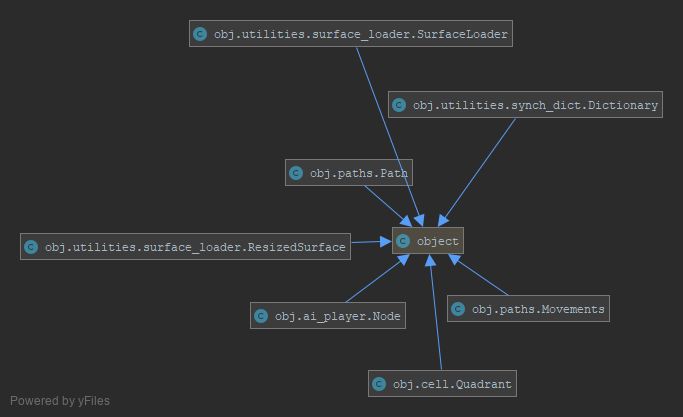


Ilustración - Diagrama UML del módulo de estructuras

Tabla - Clases clave del módulo de estructuras

#### Módulo de utilidades

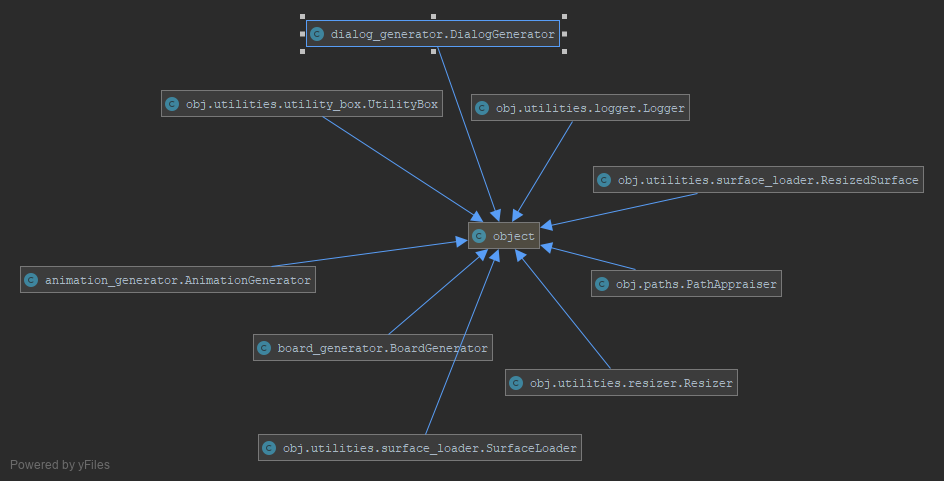


Ilustración - Diagrama UML del módulo de utilidades

Tabla - Clases clave del módulo de utilidades

Los conjuntos de métodos que juntan estas clases son imprescindibles a lo largo de toda la aplicación, otras muchas clases y objetos se apoyan en las dependencias resultantes.

## ESTADÍSTICAS

### ESTADÍSTICAS DE ARCHIVOS LOCALES

Hay que tener en cuenta que algunos de los archivos no tienen documentación interna, por considerarse innecesaria o de baja prioridad respecto al resto de aspectos de la aplicación.

El proyecto contiene 12858 líneas de código Python, de las cuales 7582 son código puro, 4271 documentación interna, y 1005 líneas están en blanco al usarse en el estilo de los archivos. 34 ficheros.

Como adición tenemos el servicio de la tabla de servidores, escrito en *javascript*, con 106 líneas, 96 de código y 10 en blanco.

Todo esto acaba sumando una aplicación con 12858 (12964 con JS) líneas de código de fabricación propia, sin contar librerías externas. Aunque no sea una medida totalmente fiable, nos sirve para hacernos una idea de la envergadura de la aplicación.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de fichero | Número de ficheros | Tamaño en disco | Tamaño acumulativo | % TOTAL. /ACUMULATIVO |
| Imágenes (PNG) | 791 | 67898 Kb | 67898 Kb | 54.89% /54.89% |
| Audio (OGG) | 18 | 23928 Kb | 91826 Kb | 19.34% /74.24% |
| Imágenes (GIF) | 89 | 1671 Kb | 93497 Kb | 1.35% /75.59% |
| Código python (PY) | 36 | 836 Kb | 94333 Kb | 0.67% /76.26% |
| CÓDIGO JAVASCRIPT (JS) | 111 | 550 Kb | 94883 kB | 0.44% /76.70% |
| Otros (\*) | 297 | 28806 Kb | 123689 Kb | 23.30% 100% |

Tabla – Tabla de tamaño conjunto de ficheros por tipo, ordenados por importancia

### ESTADÍSTICAS DE CÓDIGO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Módulo** | **Archivo** | **Líneas totales /de código** | **Líneas de comentarios /en blanco** | **Porcentaje**  **Total/Acum.** |
| **-** | Old\_code.py | 2126/1125 | 780/221 | -/- |
| **Lógica** | Board.py | 1405/1001 | 310/94 | 10.84%/10.84% |
| **Gráfico** | Ui\_element.py | 1381/649 | 635/97 | 10.65%/21.49% |
| **Lógica** | Players.py | 973/452 | 440/81 | 7.50%/28.99% |
| **Gráfico** | Sprite.py | 869/434 | 364/71 | 6.70%/35.69% |
| **Lógica** | Game.py | 795/560 | 195/40 | 6.13%/41.82% |
| **Red** | Network\_board.py | 682/483 | 159/40 | 5.26%/47.08% |
| **IA** | Ai\_player.py | 635/322 | 287/26 | 4.90%/51.98% |
| **Lógica** | Screen.py | 566/385 | 134/47 | 4.36%/56.34% |
| **IA** | Paths.py | 502/324 | 134/44 | 3.87%/60.21% |
| **Gráfico** | Animation\_generator.py | 464/219 | 216/29 | 3.58%/63.79% |
| **Utilidades** | Utility\_box.py | 426/254 | 141/31 | 3.29%/67.08% |
| **Lógica** | Cell.py | 423/275 | 98/50 | 3.26%/70.34% |
| **Lógica** | Main.py | 422/249 | 146/27 | 3.25%/73.59% |
| **Gráfico** | Animation.py | 396/216 | 142/38 | 3.05%/76.64% |
| **Gráfico** | Board\_generator.py | 381/223 | 133/25 | 2.94%/79.58% |
| **Estructuras** | Surface\_loader.py | 340/124 | 176/40 | 2.62%/82.20% |
| **Red** | Board\_server.py | 297/195 | 83/19 | 2.29%/84.49% |
| **Utilidades** | Settings.py | 253/183 | 44/26 | 1.95%/86.44% |
| **Lógica** | Menu.py | 199/123 | 64/12 | 1.53%/87.97% |
| **Utilidades** | Logger.py | 133/77 | 41/15 | 1.02%/88.99% |
| **Gráfico** | Polygons.py | 129/69 | 43/17 | 1.00%/89.99% |
| **Lógica** | Tutorial\_board.py | 125/61 | 51/13 | 0.96%/90.95% |
| **Lógica** | Decorators.py | 113/66 | 38/9 | 0.87%/91.82% |
| **Gráfico** | Dialog\_generator.py | 112/51 | 54/7 | 0.86%/92.68% |
| **Gráfico** | Counter.py | 112/61 | 34/17 | 0.86%/93.54% |
| **Gráfico** | Resizer.py | 106/54 | 47/5 | 0.82%/94.36% |
| **Gráfico** | Help\_dialogs.py | 106/99 | 0/7 | 0.82%/95.18% |
| **Lógica** | Dice.py | 103/89 | 3/11 | 0.79%/95.97% |
| **Estructuras** | Synch\_dict.py | 87/44 | 32/11 | 0.67%/96.64% |
| **Gráfico** | Strings.py | 86/62 | 8/16 | 0.66%/97.30% |
| **Utilidades** | Exceptions.py | 85/58 | 0/27 | 0.65%/97.95% |
| **Red** | Ip\_parser.py | 66/45 | 15/6 | 0.51%/98.46% |
| **Utilidades** | Pygame\_test.py | 49/44 | 0/5 | 0.38%/98.84% |
| **Utilidades** | Memory\_checker.py | 23/19 | 3/1 | 0.18%/99.02% |
| **Gráfico** | Colors.py | 14/12 | 1/1 | 0.10%/99.12% |

Tabla – Tabla de ficheros .py, ordenados por líneas de código

Se han incluido en el módulo de utilidades los ficheros que contienen variables de configuración o información en texto plano.

El acumulativo final no es 100% ya que no se incluye el script con **JS y Express**.

A su vez, obtenemos la siguiente tabla de las líneas totales de cada módulo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Módulo | Número de ficheros | Líneas totales | lineas acumulativo | % TOTAL. /ACUMULATIVO |
| lógica | 9 | 5124 | 5124 | 39.85%/39.85% |
| gráfico | 12 | 4156 | 9280 | 32.32%/72.17% |
| ia | 2 | 1137 | 10417 | 8.84%/81.01% |
| RED | 3 | 1045 | 11462 | 8.13%/89.14% |
| utilidades | 6 | 969 | 12431 | 7.54%/96.68% |
| estructuras | 2 | 427 | 12858 | 3.32%/100% |

Tabla – Tabla de los módulos conformantes de la aplicación y sus estadísticas

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Módulo** | **Archivo** | **Líneas totales /de código** | **Líneas de comentarios /en blanco** | | **Porcentaje**  **Módulo/Acum.** | |
| **Lógica** | Board.py | 1405/1001 | 310/94 | | 27.42%/27.42% | |
| **Lógica** | Players.py | 973/452 | | 440/81 | | 18.99%/46.41% |
| **Lógica** | Game.py | 795/560 | 195/40 | | 15.52%/61.93% | |
| **Lógica** | Screen.py | 566/385 | 134/47 | | 11.05%/72.98% | |
| **Lógica** | Cell.py | 423/275 | 98/50 | | 8.26%/81.24% | |
| **Lógica** | Main.py | 422/249 | 146/27 | | 8.24%/89.48% | |
| **Lógica** | Menu.py | 199/123 | 64/12 | | 3.88%/93.36% | |
| **Lógica** | Tutorial\_board.py | 125/61 | 51/13 | | 2.44%/95.80% | |
| **Lógica** | Decorators.py | 113/66 | 38/9 | | 2.20%/98.00% | |
| **Lógica** | Dice.py | 103/89 | 3/11 | | 2.00%/100% | |
| **Gráfico** | Ui\_element.py | 1381/649 | 635/97 | | 33.23%/33.23% | |
| **Gráfico** | Sprite.py | 869/434 | 364/71 | | 20.91%/54.14% | |
| **Gráfico** | Animation\_generator.py | 464/219 | 216/29 | | 11.16%/65.30% | |
| **Gráfico** | Animation.py | 396/216 | 142/38 | | 9.53%/74.83% | |
| **Gráfico** | Board\_generator.py | 381/223 | 133/25 | | 9.17%/84.00% | |
| **Gráfico** | Polygons.py | 129/69 | 43/17 | | 3.10%/87.10% | |
| **Gráfico** | Dialog\_generator.py | 112/51 | 54/7 | | 2.69%/89.79% | |
| **Gráfico** | Counter.py | 112/61 | 34/17 | | 2.69%/92.48% | |
| **Gráfico** | Resizer.py | 106/54 | 47/5 | | 2.55%/95.03% | |
| **Gráfico** | Help\_dialogs.py | 106/99 | 0/7 | | 2.55%/97.58% | |
| **Gráfico** | Strings.py | 86/62 | 8/16 | | 2.07%/99.65% | |
| **Gráfico** | Colors.py | 14/12 | 1/1 | | 0.35%/100% | |
| **IA** | Ai\_player.py | 635/322 | 287/26 | | 55.85%/55.85% | |
| **IA** | Paths.py | 502/324 | 134/44 | | 44.15%/100% | |
| **Red** | Network\_board.py | 682/483 | 159/40 | | 65.26%/65.26% | |
| **Red** | Board\_server.py | 297/195 | 83/19 | | 28.42%/93.68% | |
| **Red** | Ip\_parser.py | 66/45 | 15/6 | | 6.32%/100% | |
| **Utilidades** | Utility\_box.py | 426/254 | 141/31 | | 43.96%/43.69% | |
| **Utilidades** | Settings.py | 253/183 | 44/26 | | 26.21%/69.90% | |
| **Utilidades** | Logger.py | 133/77 | | 41/15 | | 13.78%/83.68% |
| **Utilidades** | Exceptions.py | 85/58 | 0/27 | | 8.80%/92.48% | |
| **Utilidades** | Pygame\_test.py | 49/44 | 0/5 | | 5.07%/97.55% | |
| **Utilidades** | Memory\_checker.py | 23/19 | 3/1 | | 2.45%/100% | |
| **Estructuras** | Surface\_loader.py | 340/124 | 176/40 | | 79.63%/79.63% | |
| **Estructuras** | Synch\_dict.py | 87/44 | 32/11 | | 20.37/100% | |

Tabla – Tabla de ficheros .py, ordenados por módulo

Tabla - Gráfico de porcentajes de cada módulo

### Estructuras de datos

Tabla – Estructuras de datos utilizadas y sus propósitos

Las matrices que no son *LUTs*, son realmente grafos, uno para los **caminos conectados directamente** (con distancia 1 entre casillas), y otro para las **distancias entre casillas** (número de espacios en un camino directo entre casillas).

Para las colisiones se usan las **distancias euclídeas** en algunos casos.

En diccionarios se guardan las **puntuaciones de fitness** realizadas en el turno concreto de la partida, para gastar ciclos de reloj devolviendo la misma información. Después de cada turno, al haber cambiado la situación, se limpia la estructura.

También guardan la **situación actual del mapa** (la *key* es el índice de la casilla, y el objeto es una instancia de casilla simplificada, que tiene solo la información que nos interesa en esta estructura (objeto *Path*))

**Cuadrantes**, son los conjuntos de recuadros de cada jugador, y están divididos según los puntos cardinales. (hasta 4 jugadores). Algunos se solapan, pero las zonas solapadas solo se usan cuando no le correspondan a otro jugador.



**CUADRANTE 2**

**CUADRANTE 4**

**CUADRANTE 3**

**CUADRANTE 1**

Ilustración – Ejemplo de cuadrantes en un tablero

Tablas de hashes**,** funcionan guardando un **hash** por imagen como **key**. Al cargar otro objeto, se comprueba primero si ya ha sido procesado anteriormente (si el hash correspondiente existe).

Dicho **hash** es creado a partir de la ruta de la imagen, si se carga sin modificar, o usando la ruta y los distintos parámetros de redimensionado, si es procesada y se le cambia el tamaño. En los caminos, el hash es según las restricciones de movimiento.

## ALGORITMIA

Cada conjunto de métodos es descrito brevemente, realizado en pseudocódigo, y sus resultados mostrados.

### PROCESADO INICIAL, LUTS

|  |
| --- |
| GENERADOR\_MATRIZ\_EUCLIDEA\_CUADRADA(TAMAÑO):  para x desde 0 hasta tamaño+1:  para y desde x hasta tamaño+1:  DIST\_EUCLIDEA = RAIZ\_CUADRADA(x\*x + y\*y)  matriZ[x][y] = DIST\_EUCLIDEA  matriZ[y][x] = DIST\_EUCLIDEA  retornar matriZ |

Tabla – Pseudocódigo de distancias euclídeas

|  |  |  |
| --- | --- | --- |
| Método | Tiempo de cómputo | Ahorro medio |
| Generación inicial | 33.03ms + 0.0011ms/acceso | 44ms/turno 🡪 59.3% |
| Generación en tiempo real | 0.0027ms/movimiento | 108ms/turno 🡪 -145% |

Tabla - Comparación de rendimiento usando un LUT para distancias euclídeas (Con tamaño 300\*300)

Las distancias euclídeas se usan para las colisiones, ergo si queremos saber la distancia de un punto *(pixel X, pixel Y)* con otro cualquiera, debemos realizar una raíz cuadrada de la diferencia de píxeles en ambos ejes.

Se usa de tamaño 300 (300\*300 = 90000) ya que, si los objetos están entre sí separados en cualquier eje más de esta distancia, será seguro que no están en contacto. Hay que tener en cuenta que estos tiempos son en condiciones óptimas.

Sin generar previamente estos cálculos, en cada movimiento de ratón (por un píxel) se debería comprobar la distancia con todos los objetos que usen este método para detectar colisiones, unas 40000 veces/turno.

Nota: Para calcular los tiempos de ejecución de fragmentos pequeños de código, que suelen tardar mucho menos de 1 milisegundo, se usa la librería timeit con la función del mismo nombre, poniendo el código como un callback

|  |
| --- |
| timeit.timeit(lambda: math.sqrt(240\*240 + 255\*255) |
| timeit.timeit(lambda: array[0]) |

Tabla - Código usado para sacar el tiempo gastado en un acceso y en el cálculo de una raíz

|  |
| --- |
| CALCULAR\_VALORES\_FITNESS(ÍNDICE\_CASILLA):  SI SELF.FITNESS[ÍNDICE\_CASILLA] EXISTE:  RETORNAR  PARA ÍNDICE\_DESTINO EN ÍNDICE\_CASILLA:  RESULTADO = PUNTUACIÓN\_MOVIMIENTO(ÍNDICE\_CASILLA, ÍNDICE\_DESTINO)  LISTA\_PUNTUACIONES[DESTINO\_POSIBLE] = RESULTADO  SELF.FITNESS[ÍNDICE\_CASILLA] = LISTA\_PUNTUACIONES |

Tabla - Pseudocódigo de cálculo de puntuaciones de movimiento en el tablero

|  |  |  |
| --- | --- | --- |
| Método | Tiempo de cómputo (3 selecciones misma casilla) | Ahorro medio (3 selecciones misma casilla) |
| Generación inicial | (80\*1+3\*2) = **86ms** | **180%** |
| Generación en cada selección | (80\*3) = **240ms** | **0%** |

Tabla - Comparación de rendimiento usando un LUT para almacenar las puntuaciones en cada turno

En cada turno el atributo del tablero **SELF.FITNESS** se vacía, ya que los resultados calculados no sirven para el siguiente turno/jugador.

Dentro de un mismo turno, tan sólo se calculan cuando se selecciona un personaje en una casilla concreta por primera vez en ese turno.

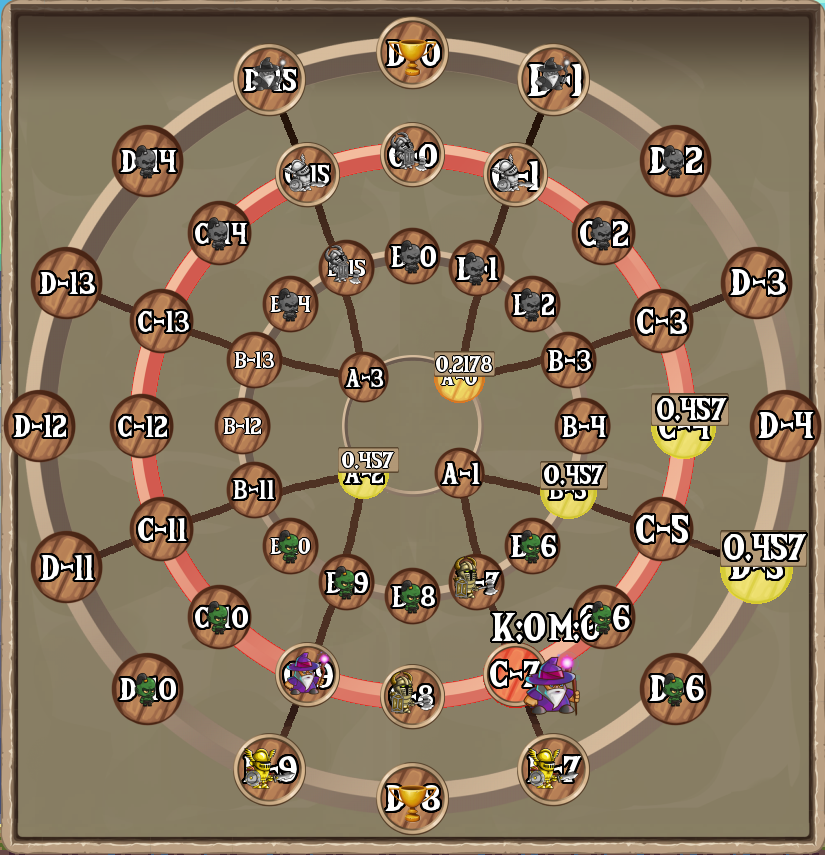


Ilustración – Ejemplo de puntuaciones ‘fitness’ en el tablero

### CAMINOS

|  |
| --- |
| GENERACIÓN\_MOVIMIENTOS\_EN\_TABLERO(TABLERO, SUBTIPOS\_PERSONAJES):  PARA TIPO\_DE\_PJ EN SUBTIPOS\_PERSONAJES:  SI TIPO\_DE\_PJ.HASH EXISTE EN SELF.CAMINOS:  CONTINUAR EN SIGUIENTE ITERACIÓN  DICCIONARIO TODOS\_LOS\_CAMINOS = {}  PARA CASILLA EN TABLERO.CASILLAS:  CAMINOS = CAMINOS\_RESULTANTES(CASILLA.ÍNDICE)  TODOS\_LOS\_CAMINOS[CASILLA.ÍNDICE] = CAMINOS  SELF.CAMINOS[TIPO\_DE\_PJ.HASH] = TODOS\_LOS\_CAMINOS |

Tabla - Lut inicial para los caminos de un tablero

|  |  |  |
| --- | --- | --- |
| Método (1000 movimientos) | Tiempo de cómputo | Ahorro medio |
| Generación inicial | (150\*1+ 1\*1000) = **1150ms** | **335%** |
| Generación en cada selección | (5\*1000) = **5000ms** | **0%** |

Tabla - Comparación de rendimiento usando un LUT para almacenar los caminos posibles

Por supuesto el pseudocódigo de este algoritmo está MUY simplificado. En realidad, lo que ocurre es una generación por demanda, o lazy.

La estructura de caminos está inicialmente vacía. Al seleccionar un personaje en el tablero, se comprueba si este personaje tiene los caminos correspondientes generados y guardados (comparando el hash). Si no los tiene, se generan esa única vez. Si los tiene, se devuelven los caminos posibles.

Para comparar hashes, se ha creado un objeto de tipo Restricción, que contiene las restricciones de desplazamiento que tiene un tipo de personaje. Éste es el objeto que se usa para el cálculo del hash.

Ilustración – Atributos de un objeto Restricción de un personaje

Estas rutas que hemos descrito serían las rutas en un tablero vacío. Después se filtran mirando que casillas tienen un aliado en ellas, que recorrido contiene otro personaje en la mitad del mismo, y/u otros parámetros.

### CONTROL DE IMÁGENES

|  |
| --- |
| cargar\_imagen(ruta\_archivo, parámetros\_de\_redimensionado):  SI parámetros\_de\_redimensionado NO ES NULL:  hash = (ruta\_archivo, parámetros\_de\_redimensionado).hash  SI No:  hash = ruta\_archivo.hash  SI HASH EXISTE EN SELF.imagenes:  Retornar SELF.imagenes[HASH]  imagen = cargar\_imagen(ruta\_archivo)  SI parámetros\_de\_redimensionado NO ES NULL:  imagen = redimensionar\_imagen(imagen, parámetros\_de\_redimensionado)  SELF.imagenes[HASH] = imagen |

Tabla – Pseudocódigo de la estructura LUT de imágenes cargadas

|  |  |  |  |
| --- | --- | --- | --- |
| Resolución | Carga Memoria (LUT imágenes) | Carga Memoria (normal) | Ahorro |
| 1920\*1080 | 1115 MB | 1790 MB | 61% |

Tabla – Ejemplo de uso de memoria según la técnica en uso

La tabla completa con las resoluciones la encontramos en el apartado siguiente.

Cada vez que un objeto de tipo Surface o Imagen necesita ser cargado en memoria, ya sea para una instancia de Sprite concreta u otro propósito, se realiza mediante el método arriba mostrado.

De esta manera evitamos guardar imágenes duplicadas en memoria, dibujando una sola imagen en las posiciones que cada objeto requiera.

El programa tiene dos tablas de hashes que comprueba, una para las imágenes sin modificación, y otra para las modificadas. Las últimas, con este propósito, usan una clase llamada ResizedSurface, con los parámetros de alteración de la imagen.

El Hash, en este segundo caso, se produce mediante el conjunto de dichos parámetros.

Ilustración - Atributos de un objeto ResizedSurface de una imagen

### THREAD POOLING

Debido a que el proceso inicial de carga y generación de elementos tiene una gran carga paralelizada, en la mayoría de casos se tienen bastante más hilos que núcleos lógicos tiene nuestro procesador.   
Para solucionar esto, y para ahorrar costes de creación de hilos, se ha implementado un sistema de asignación de tareas a hilos ociosos, creados previamente, lo que se llama tradicionalmente en inglés Thread Pooling.

Si

Ilustración - Flujo de paso de las tareas mediante Thread Pooling

Este sistema se complementa con la creación de hilos sin limitación, ya que en determinadas situaciones no puedes asegurarte de, o esperar a que haya un hilo ocioso dispuesto a ejecutar nuestra tarea. Sobre todo, por el bien de la fluidez de la aplicación y el feedback que debe recibir el usuario por sus acciones.

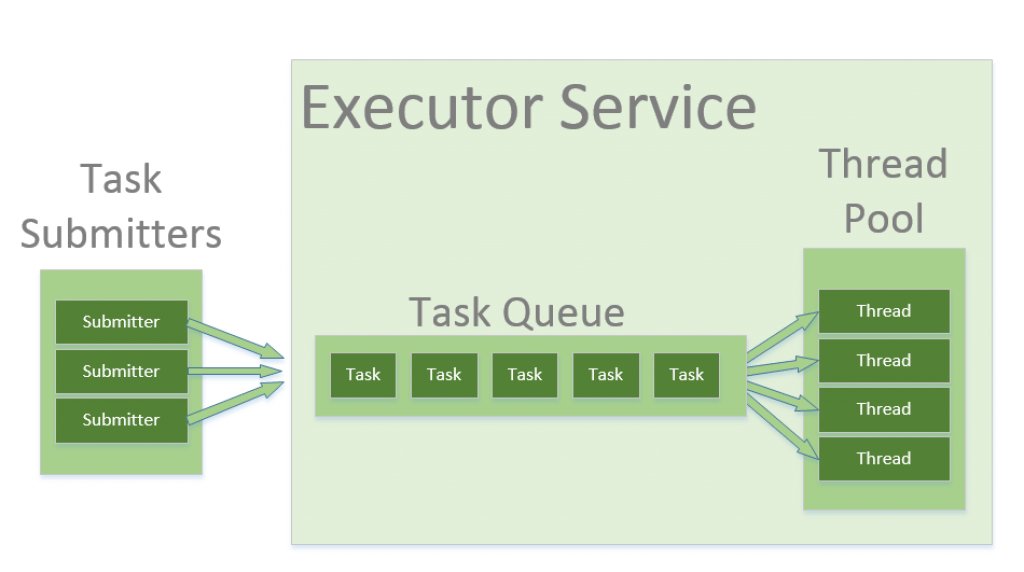


Ilustración – Representación gráfica del proceso (Agrupamiento de hilos)

### Gráficas y datos finales

#### Profiler

Debido a la naturaleza del proyecto, sobre todo en las etapas iniciales, es necesario monitorizar cuanta memoria consume el programa en las situaciones más exigentes, para detectar posibles errores de memoria (MemoryError en **Python**) y métodos ineficientes.

La librería memory-profiler (<https://pypi.org/project/memory-profiler/>) es la elegida para ayudarnos con esta tarea. Proporciona lecturas del espacio que ocupa el programa en RAM, en cada línea y después de cada llamada a un método.

Las comparaciones de carga en memoria según métodos de las secciones anteriores, se han realizado con esta librería.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resolución | Uso máx. Memoria (LUT) | Uso máx. Memoria (Normal) | Tiempo Carga Inicial/Tablero (LUT) | Tiempo Carga  Inicial/Tablero (Normal) |
| 640\*360 | 830 MB | 1630 MB | 5410ms/25800ms | 6900ms/125680ms |
| 848\*480 | 950 MB | 1715 MB | 6400ms/27500ms | 7500ms/133500ms |
| 1024\*576 | 970 MB | 1760 MB | 7500ms/27900ms | 8900ms/124500ms |
| 1280\*720 | 1070 MB | 1880 MB | 9300ms/30800ms | 10500ms/131100ms |
| 1366\*768 | 1120 MB | 1950 MB | 10000ms/31100ms | 11000ms/132000ms |
| 1600\*900 | 1245 MB | 2100 MB | 12100ms/34200ms | 12500ms/132300ms |
| 1920\*1080 | 1370 MB | 2280 MB | 16500ms/38900ms | 20100ms/131500ms |

Tabla – Comparación de gasto de memoria y tiempo para todas las resoluciones

Todas las pruebas para la tabla superior se han realizado con limitación del tamaño de algunas imágenes, como los personajes, ya que dichas superficies nunca superarán ciertas proporciones en entornos reales (Debido a la proporción con la pantalla).

Las cargas del teclado se realizan siempre en el ajuste más exigente de **4 jugadores**, cada uno con sus caracteres, en un tablero clásico (64 casillas).

Un ejemplo de la máxima resolución sin dicha limitación:

|  |  |  |  |
| --- | --- | --- | --- |
| Gasto Memoria (LUT) | Gasto Memoria (Normal) | Tiempo Carga Inicial/Tablero (LUT) | Tiempo Carga  Inicial/Tablero (Normal) |
| 1550 MB | **2450 MB** | 16600ms/43300ms | 16886ms/132200ms |

Tabla – Gasto de memoria con el tamaño de las superficies deslimitado (Resolución 1920\*1080)

|  |  |  |  |
| --- | --- | --- | --- |
| Resolución | Ahorro memoria | Ahorro tiempo  Carga Inicial | Ahorro tiempo Carga Tablero |
| 640\*360 | 96% | 28% | 387% |
| 848\*480 | 80% | 17% | 385% |
| 1024\*576 | 81% | 19% | 346% |
| 1280\*720 | 76% | 13% | 325% |
| 1366\*768 | 74% | 10% | 324% |
| 1600\*900 | 69% | 3% | 287% |
| 1920\*1080 | 66% | 22% | 238% |

Tabla - Ahorro de memoria y tiempo para todas las resoluciones, usando LUTs

El ahorro de tiempo tan acusado tiene como razón el reciclado de las imágenes ya cargadas.

Cuando una instancia carga una imagen, si ésta ha sido cargada antes, al consigue ya procesada con un simple acceso. En caso contrario debería leer de nuevo el archivo del disco, procesarlo y asignarlo.

Como apunte, aunque el punto álgido de memoria ocupada esté en el rango de los gigabytes, esto solo ocurre en el proceso de carga. En todas las pruebas baja hasta incluso la mitad durante la partida.

#### Pruebas de rendimiento

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Equipo | Procesador | Consumo | [Rendimiento](https://www.cpubenchmark.net/cpu_list.php) | Precio (Est.) | Relación |
| MSI | I7 7700HQ | 45W/230W | 8806 pts | 1000€ | 8.8pts/€ |
| Thinkpad | I5 6200U | 15W/45W | 4020 pts | 400€ | 10.0pts/€ |
| Lenovo | E1 6010 | 10W/45W | 862 pts | 180€ | 4.8pts/€ |
| Intel Stick | Atom Z3735F | 4.4W/10W | 908 pts | 60€ | 15.1pts/€ |

Tabla - Comparativa de equipos de prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Equipo | Resolución | Carga Inicial /Tablero 2-4 jug  (ms) | FPS en Menús  (Media, MAX 120 FPS) | FPS tablero 2 / 4 jugadores |
| MSI | 640\*360 | 5900/17660/26880 | 115 FPS | 115 FPS/115 FPS |
| MSI | 1280\*720 | 9600/21430/30080 | 100 FPS | 60FPS/60 FPS |
| MSI | 1920\*1080 | 15800/29500/39500 | 60 FPS | 30 FPS/30 FPS |
| Thinkpad | 640\*360 | 6600/21400/30400 | 110 FPS | 110 FPS/110 FPS |
| Thinkpad | 1280\*720 | 11080/26700/36700 | 110 FPS | 50 FPS/50 FPS |
| Thinkpad | 1920\*1080 | 20400/42400/54000 | 50 FPS | 25 FPS/25 FPS |
| Lenovo G50 | 640\*360 | 27200/91700/119000 | 100 FPS | 45 FPS/ |
| Lenovo G50 | 1280\*720 | 45000/101500/126600 | 30 FPS | 15 FPS/15 FPS |
| Lenovo G50 | 1920\*1080 | 60000/121000/154700 | 15 FPS | 7 FPS / 7 FPS |
| Intel Stick | 640\*360 | 30600/100100/137500 | 85 FPS | 30 FPS/30 FPS |
| Intel Stick | 1280\*720 | 51200/123700/189900 | 20 FPS | 10 FPS/10 FPS |
| Intel Stick | 1920\*1080 | 92400/-/- | 10 FPS | -/- |

Tabla – Pruebas de rendimiento en los distintos entornos

|  |  |  |  |
| --- | --- | --- | --- |
| Equipo | Algoritmo | Rendimiento | Tiempo |
| MSI | Alfa Beta | 36570 nodos | 10 segundos |
| MSI | Montecarlo | 480 iteraciones | 10 segundos |
| Thinkpad | Alfa Beta | 20350 nodos | 10 segundos |
| Thinkpad | Montecarlo | 305 iteraciones | 10 segundos |
| Lenovo G50 | Alfa Beta | 6140 nodos | 10 segundos |
| Lenovo G50 | Montecarlo | 59 iteraciones | 10 segundos |
| Intel Stick | Alfa Beta | 2730 nodos | 10 segundos |
| Intel Stick | Montecarlo | 30 iteraciones | 10 segundos |

Tabla – Comparación de rendimiento de las IAs (Media) (En resolución 640x360)

#### Conclusiones de pruebas reales

La librería utilizada, en su estado base, no dispone de la lógica para utilizar la tarjeta gráfica dedicada. Debido a esto, la máquina más potente (y la única que posee un procesador gráfico no integrado en el procesador), no se desmarca tanto como debiera del resto, juzgando por el precio.

Al tener los modelos **MSI** y **Thinkpad** procesadores con gráficos integrados parecidos (Intel HD 630 vs Intel HD 520), obtienen tasas medias de FPS cercanas.

La diferencia entre la capacidad de cómputo de los procesadores se hace evidente, tanto en los tiempos de carga iniciales y del tablero, como en el rendimiento de los algoritmos heurísticos en un intervalo de tiempo definido.

Tabla – Gráfica de comparación entre equipos (FPS y tiempos de carga)

Tabla - Gráfica de comparación entre equipos (Rendimiento heurísticas)

## Sistema de servidor y cliente

Tenemos cuatro situaciones posibles:

1.- Que un usuario quiera crear un **servidor de modo privado**. Para conectarse a él hace falta saber la dirección IP y puerto del host.

2.- Usuario que inicia un **servidor de modo público**. Para conectarse a él tan solo hay que seleccionarlo de la tabla de servidores online.

3.- Cliente que **conecta a un servidor privado**.

4.- Cliente que **se une a un servidor público**.

El ordenador que inicie el servidor también crea un cliente, el llamado “Maestro” o “Host” de una partida. Será éste el que decida parámetros como el número total de jugadores, cuántos jugadores estarán controlados por la máquina, o el tamaño del tablero.

También deberá encargarse de hacer llegar esta información a los otros participantes que se conecten a su servidor, así como de procesar, controlar y mover los agentes jugadores no humanos.

### ESQUEMA DEL MODELO USADO

Aquí se enseñan dos modelos, dependiendo de si el servidor está en modo público o privado. La leyenda se ve en la siguiente tabla:

Tabla – Leyenda de los esquemas de red

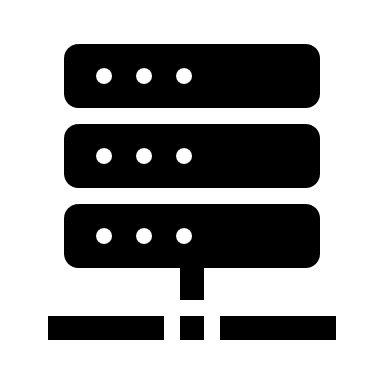
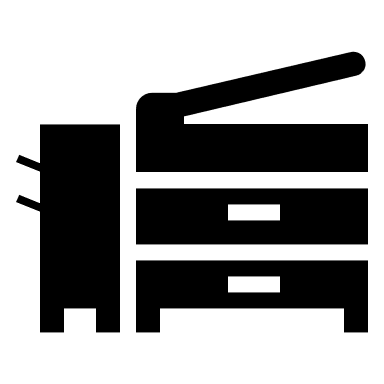


Ilustración – Esquema de comunicación en una conexión a un servidor privado



Registra el servidor

Recibe los servidores

Ilustración – Esquema de comunicación en una conexión a un servidor privado

### Tabla de servidores públicos (API)

Si es público, se aprecia el paso adicional de enviar la información de conexión del servidor a otra máquina externa.

Dicha máquina, ajena al resto del caos, se ocupa únicamente de recibir que máquinas están online de forma pública, y de enviar esta información a los usuarios interesados.

Para sufragar la implementación de este sistema, he realizado una API REST, usando el entorno de ejecución **Node** (lenguaje de programación, **Javascript**), y la librería Express.

La API tiene cuatro puntos accesibles (**endpoints**):

Ilustración – Todos los puntos de acceso del servidor de Express

Cada punto de acceso tiene un middleware configurado para rechazar peticiones que no contengan todos los parámetros necesarios.

En el entorno real, existen dos maneras de tener este sistema encendido y funcionando:

1.- De manera local en nuestra máquina. Útil solo para pruebas en la red local de nuestra casa.

2.- Mediante una **máquina virtual** ejecutándose en **la nube de servidores de Amazon (AWS)**, dentro de la cual se encuentra el programa de ***Node***.

Para que la segunda manera sea más útil, debemos tener una dirección de dominio para nuestra API, ya que las te asigna Amazon en sus máquinas virtuales no son demasiado intuitivas o cómodas.

|  |
| --- |
| ec2-34-205-125-210.compute-1.amazonaws.com |

Tabla – Ejemplo de dirección DNS de máquina virtual de Amazon.

Noip es la solución a este problema, emparejando el nombre de dominio que deseemos con la IP que le digamos.

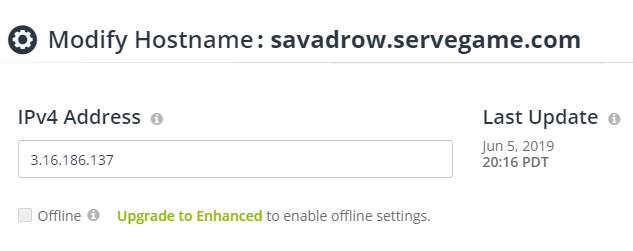


Ilustración – Panel de ajustes de **Noip**

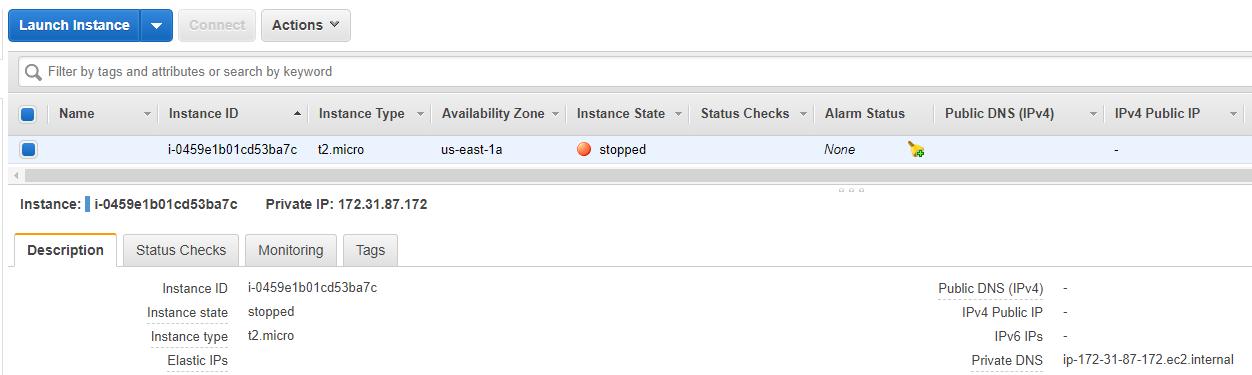


Ilustración – Panel de control de instancias en el servicio web de Amazon (AWS)

### TABLA DE MENSAJES

**Conexión/Sincronización**

|  |  |  |  |
| --- | --- | --- | --- |
| Mensaje | Origen | Destino | Función |
| host | Cliente (Todos) | Servidor | Registro de conexión |
| start | Servidor | Cliente (Todos) | Comienzo de partida |
| ready | Cliente (Todos) | Servidor | Tablero creado |
| pause | Cliente (Todos) | Cliente (Todos) | Pausar partida |
| params | Cliente (Host) | Cliente (Invitado) | Parámetros partida |
| players | Cliente (Invitado) | Servidor | ¿Cuantos jugadores? |
| start\_dice | Cliente (Todos) | Servidor | Valor tirada inicial |
| keep\_alive | Cliente (Todos) | Servidor | Mantener la conexión |
| disconnect | Cliente (Todos) | Servidor | Desconectar cliente |
| players\_data | Cliente (Host) | Cliente (Invitado) | Datos de jugadores |
| characters\_data | Cliente (Host) | Cliente (Invitado) | Datos de personajes |

Tabla – Mensajes usados en la conexión y sincronización de usuarios

**Partida en curso**

|  |  |  |  |
| --- | --- | --- | --- |
| Mensaje | Origen | Destino | Función |
| swap | Cliente (Todos) | Cliente (Todos) | Promoción personaje |
| admin | Cliente (Todos) | Cliente (Todos) | Modo admin activado |
| end\_turn | Cliente (Todos) | Cliente (Todos) | Final de turno |
| turncoat | Cliente (Todos) | Cliente (Todos) | Modo traidor True |
| dice\_value | Cliente (Todos) | Cliente (Todos) | Valor de tirada |
| move\_character | Cliente (Todos) | Cliente (Todos) | Movimiento personaje |
| drop\_character | Cliente (Todos) | Cliente (Todos) | Colocar pj en casilla |
| lock\_characters | Cliente (Todos) | Cliente (Todos) | Bloquear personajes |
| mouse\_position | Cliente (Todos) | Cliente (Todos) | Movimiento cursor |
| update\_character | Cliente (Todos) | Cliente (Todos) | Promoción personaje |

Tabla – Mensajes utilizados en el transcurso de una sesión de juego

### FLUJO EN UNA GENERACIÓN/CONEXIÓN

|  |
| --- |
| 1.- El Host crea un servidor que aloje la partida y haga de intermediario entre los usuarios. |
| ServerComputer |

|  |
| --- |
| 1.2.- Host registra el servidor creado en el servicio externo con la tabla de servidores públicos. |
| ServerComputerPhotocopier |

|  |
| --- |
| 1.3.- Clientes reciben la tabla de servidores públicos, y eligen uno del cual consiguen los datos de conexión. |
| LaptopPhotocopierLaptopLaptop |

|  |
| --- |
| 2.- Clientes se envían su conexión al servidor y éste la acepta/rechaza (El host ya es cliente de esa partida). |
| LaptopLaptopLaptopServer |

|  |
| --- |
| 3.- Host genera la partida, y envía todos los parámetros una vez creada (jugadores, personajes, atributos del tablero). |
| ServerComputer |

|  |
| --- |
| 4.- Los clientes invitados reciben los parámetros del tablero, los crean, generan jugadores y personajes vacíos, y los rellenan al recibir los datos del host. Ya están todos los usuarios listos. |
| LaptopLaptopLaptopServer |

Tabla – Proceso de generación de una partida en un entorno de red (Clientes y Servidor)

Inteligencia Artificial

### Algoritmo de fitness

En el método más completo, dividimos la puntuación de un movimiento concreto en 3 componentes:

1.- Puntuación de peligro: Se basa en la diferencia entre la posibilidad de ser capturada que tenía la pieza en la posición inicial y en el destino.

Este valor tiene un multiplicador llamado Multiplicador de peligro, que depende casi exclusivamente de la cantidad de piezas del mismo tipo que tenga el jugador. Es decir, se premia la variedad de personajes.

|  |
| --- |
|  |

Ecuación – Fórmula del multiplicador de peligro

|  |
| --- |
|  |

Ecuación - Fórmula de la puntuación de peligro

2.- Puntuación de captura: Compara el valor del personaje del jugador actual, y el valor del personaje enemigo que es capturado en el destino.

|  |
| --- |
|  |

Ecuación – Fórmula de la puntuación de captura

3.- Puntuación de Cebo: Comprueba que jugadores enemigos pueden moverse al destino, y capturar este personaje después del movimiento, y a su vez, cuantos y que aliados tienen la posibilidad de hacer lo mismo.

De este modo, es posible mover un peón, que sirve de señuelo para que una pieza enemiga de más valor se mueva al destino, solo para ser capturada por una segunda pieza aliada. Esta parte del algoritmo es ignorada en el método rápido.

|  |
| --- |
|  |

Ecuación – Fórmula de la puntuación de cebo

|  |
| --- |
|  |

Ecuación – Fórmula final del algoritmo de fitness

### HEURÍSTICAS

#### Poda Alfa Beta

La poda alfa beta es un algoritmo mejorado basado en MiniMax.

La búsqueda **MiniMax** es primero en profundidad, y se basa en que la elección en nuestro turno siempre maximizará nuestro beneficio, ocurriendo lo inverso en el turno del contrincante.

El problema**,** es que el número de estados a explorar es exponencial al número de movimientos, siendo inviable usarlo en la mayoría de entornos reales.

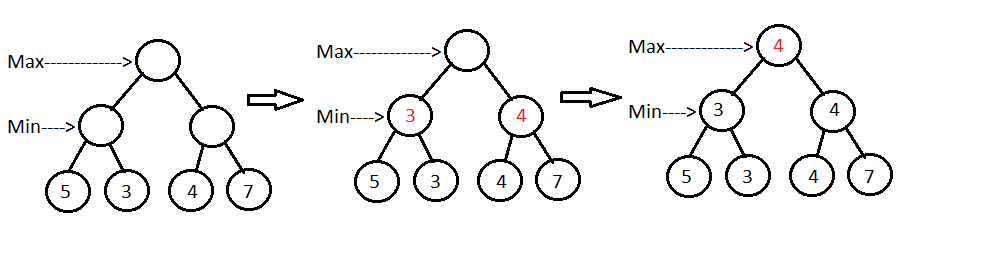


Ilustración – Ejemplo de ejecución MiniMax

Sabiendo esto, el núcleo de la poda alfa-beta se basa en la eliminación de ramas del árbol, consiguiendo así no explorar estados a los que jamás se llegaría de forma natural, ahorrando espacio y tiempo de ejecución, y reduciendo el espacio de búsqueda.

La poda de estas partes del árbol no tiene diferencia en el resultado en las comparaciones realizadas con **MiniMax**.

α es el valor de la mejor opción hasta el momento para MAX, y β es el valor de la mejor opción hasta el momento para MIN. De ahí el nombre.

Durante una ejecución se va actualizando el valor de los parámetros según se recorre el árbol. El método realizará 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.

|  |
| --- |
| Alfa\_beta(profundidad\_maxima, profundidad\_actual, α, β, tablero\_actual, etapa\_max):  SI profundidad\_actual ES profundidad\_maxima O tablero\_actual.fin:  Retornar tablero\_actual.valor  V = - ∞ SI **etapa**\_**max**, V = ∞ SI NO **etapa**\_**max**  Para movimiento en tablero\_actual.posibles\_movimientos:  tablero\_actual.simulacion(movimiento)  V’ = Alfa\_beta(profundidad\_maxima, profundidad\_actual +1, alfa, beta, tablero\_actual, not\_etapa\_max)  SI etapa\_max:  **V = MAX(V, V’)**  α = MAX(α, V’)  SI NO etapa\_max:  **V = MIN(V, V’)**  β **= MIN(**β**, V’)**  SI β < α:  BREAK AND Return V (este loop (Poda de esta rama)) |

Tabla – Pseudocódigo del algoritmo de poda alfa-beta utilizado

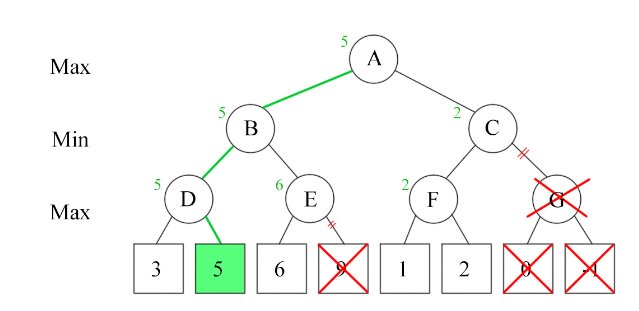
[](https://helloacm.com/the-chess-ai-model-base-or-machine-learning/)

Ilustración – Representación gráfica del árbol resultante tras una ejecución de poda alfa beta

#### Búsqueda de MonteCarlo

Con el nombre completo de árbol de búsqueda Monte Carlo (MCTS) es un algoritmo de búsqueda para procesos de toma de decisiones.

En búsquedas en árboles, siempre existe la posibilidad de que la mejor acción actual no sea realmente la acción más óptima. En estos casos es donde MCTS más brilla, ya que continua evaluando otras alternativas periódicamente.

Esto se conoce como “solución Exploración-Explotamiento”.

Aprovecha las acciones y estrategias con mejores valores hasta el momento, pero también continúa explorando el espacio local de decisiones, para comprobar si encuentra alguna alternativa que pudiera reemplazar la mejor actual.

Mientras que la exploración descubre partes inexploradas del árbol, lo cual es potencialmente beneficioso, se vuelve ineficiente en situaciones con un gran número de estados posibles, y es aquí donde se equilibra con la explotación de las soluciones ya encontradas.

La fórmula correspondiente se llama Upper Confidence Bound applied to Trees

|  |
| --- |
|  |

Ecuación – Fórmula del límite de confianza superior aplicado a árboles usado en MonteCarlo

Cada iteración de búsqueda de árbol de Monte Carlo tiene 4 etapas\_

Selección: empezar desde el nodo raíz y seleccionar nodos hijos hasta alcanzar un nodo hoja (Sin explorar).

Expansión: Si el nodo hoja no es un estado final del tablero, se crean sus nodos hijos, y elegimos entre ellos un nodo C. Los nodos hijos son cualquier movimiento válido.

Simulación: Desde el nodo elegido, simulamos movimientos hasta un estado final.

Retropropagación: Con el resultado de la fase anterior, se actualiza la información en los nodos en el camino seguido.

|  |
| --- |
| BúsquedaÁrbolMonteCarlo(nodo\_raíz):  MIENTRAS tiempo < límite:  MIENTRAS nodo\_raíz NO ES nodo\_hoja: #Selección y exploración  SI nodo\_raíz tiene hijo sin explorar:  Hoja = nodo\_raíz.hijo\_sin\_explorar  SI NO:  nodo\_raíz = nodo\_raíz.hijo\_con\_mejor\_UCBT  MIENTRAS Hoja NO ES estado\_final: #Simulación  Hoja = Movimiento\_según\_política\_de\_decisión(Hoja)  valor\_tablero\_final = Hoja.valor\_tablero  MIENTRAS Hoja NO ES nodo\_raíz: #Retroprograpagación  Hoja.valor\_total += valor\_tablero\_final  Hoja.visitas\_totales += 1  Hoja = Hoja.nodo\_padre |

Tabla – Pseudocódigo del algoritmo de búsqueda MonteCarlo utilizado

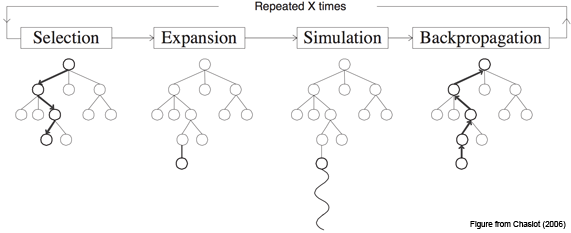


Ilustración – Representación gráfica proceso del MCTS

<https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/>

<https://en.wikipedia.org/wiki/Monte_Carlo_tree_search>

<http://mcts.ai/about/>

<https://www.chessprogramming.org/index.php?title=UCT&mobileaction=toggle_view_mobile>

<https://es.wikipedia.org/wiki/%C3%81rbol_de_b%C3%BAsqueda_Monte_Carlo>

### Decisores de movimientos

Al margen de las heurísticas, hay 3 generadores de movimientos básicas, todos basadas en menor o mayor medida, en la puntuación fitness de todos los movimientos posibles de un tablero en ese turno.

|  |  |
| --- | --- |
| Aleatorio | Se elige un movimiento al azar de todos los disponibles. |
| Aleatorio con pesos fitness | Se crean unos pesos ponderados con las puntuaciones fitness de los movimientos, y se elige un movimiento al azar teniendo en cuenta dicha ponderación. |
| Fitness puro | Se elige el movimiento con la puntuación fitness más alta.  Si hay varios con la misma puntuación, se elige uno al azar. |

Tabla – Tabla de generadores de movimientos en tablero no heurísticos

### Mejoras aplicadas

La eficacia de la poda alfa beta depende del orden en el que se examinan los sucesores, por tanto, será más eficiente si examinamos primero los mejores nodos.

Por ello, la mejora realizada ha sido la ordenación de los nodos resultantes al desplegar los movimientos posibles en el algoritmo alfa beta.

|  |
| --- |
| Alfa\_beta(profundidad\_maxima, profundidad\_actual, α, β, tablero\_actual, etapa\_max):  SI profundidad\_actual ES profundidad\_maxima O tablero\_actual.fin:  Retornar tablero\_actual.valor  V = - ∞ SI **etapa**\_**max**, V = ∞ SI NO **etapa**\_**max**  movimientos = tablero\_actual.posibles\_movimientos  fitnesses = calcular\_fitnesses(movimientos)  movimientos = movimientos.ordenar(fitnesses decrecientes)  ~~Para movimiento en tablero\_actual.posibles\_movimientos:~~  Para movimiento en movimientos:  tablero\_actual.simulacion(movimiento)  … |

Tabla – Cambios realizados en el pseudocógido de alfa beta al incluir ordenación

### Resultados algoritmos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Tabla – Comparación de decisores de movimiento

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Tabla – Comparación de rendimiento de heurísticas según tiempo

# POSIBLES AMPLIACIONES

## Tablas transposicionales

Estructuras que actúan como bases de datos de todos los procesos heurísticos realizados hasta la fecha, ayudan a acelerar este tipo de algoritmos, consiguiendo como consecuencia un espacio de búsqueda mayor en el mismo tiempo.

En la aplicación ya están implementados hashes de tableros y situaciones del mismo, por lo que esta mejora es realista.

## Redes neuronales, aprendizaje con refuerzo…

Mediante el uso de **Q-Learning** o la técnica deseada, la introducción de una o más redes neuronales puede tener como resultado la creación de un agente no humano que aprenda y mejore su estilo de juego de una manera similar a la de una persona.

Dicha ampliación requeriría bastante más tiempo para implementar, probar y comprobar, pero tiene un nivel de interés mucho más alto que la anterior.

# COSTES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Concepto | Tiempo | Precio | Total | Acumulativo |
| Elementos gráficos | **-** | **25$+25$** | **44€** | **44€** |
| Costes RRHH | **698 horas + 100 horas** | **12€/hora** | **9576€** | **9620€** |
| Costes variables (electricidad) | **798hr 🡪 PC 0.23KW**  **50hr 🡪 PC 0.045KW** | **0.14€/KWH** | **25€** | **9245€** |

Tabla – Costes finales del proyecto

|  |  |  |  |
| --- | --- | --- | --- |
| Iteración | Tiempo | Coste individual | Coste acumulativo |
| Aplicación | **307 horas** | **3684€** | **3684€** |
| Red | **124 horas** | **1488€** | **5172€** |
| Inteligencia artificial | **267 horas** | **3204€** | **8376€** |
| Documentación + Presentación + Recogida datos | **100 horas** | **1200€** | **9576€** |

Tabla – División del coste de trabajo humano según etapas

### ACLARACIONES

\*El cálculo según la electricidad es pesimista, ya que los dos equipos usados no tenían un consumo máximo de energía en todo momento.

\*El segundo equipo de menor potencia es el usado durante las pruebas de la comunicación de red.

\*Los costes humanos han sido calculados según el sueldo medio de una persona sin experiencia, pero con la formación universitaria terminada.

\*No se entiende en el entorno del cálculo de costes el coste de tiempo del proyecto como una “formación” o “periodo de prueba”, como podría pasar en el entorno empresarial real.

\*Los elementos gráficos tienen un precio conjunto debido a que se compraron con sus correspondientes licencias en dos [Humble Bundles](https://www.humblebundle.com/).

# CONCLUSIONES

## ALCANCE DE OBJETIVOS

De los tres objetivos principales, se han cumplido dos al completo, y uno parcialmente.

El **desarrollo de la** **aplicación final** como pilar para los demás sistemas **ha sido un éxito,** **superando las expectativas** y añadiendo funcionalidades extra (Como una librería de creación de elementos gráficos propia como botones y diálogos), y estructuras de aumento de eficiencia (Tablas LUT, reusado de imágenes).

**Completado al 100% también el sistema de conexión y comunicación** entre usuarios de la aplicación mediante **sockets**, que permite el envío y recepción de mensajes JSON para sincronizar ambos clientes durante una partida, o para informar de un estado adverso durante la ejecución.

En la investigación y creación de un **algoritmo de automatización de acciones**, hemos tenido que **renunciar al objetivo inicial**, que era la implementación de un algoritmo de IA, que aprendiese y evolucionase mediante aprendizaje no supervisado.

Se han realizado en su lugar 3 sistemas simples que usan un algoritmo de puntuación de movimientos, y 2 que funcionan mediante algoritmos heurísticos. Como jugadores de la máquina sirven, pero no aprenden como resultado de la participación en diversos tableros.

## CONJUNTO: RESULTADO FINAL

Como opinión personal, la adquisición de conocimientos sobre Inteligencia Artificial servirá de base para el pulido de habilidades en este campo, así como la implementación de las heurísticas. No se han cumplido todos los objetivos iniciales en este ámbito debido a las restricciones, pero el resultado, tanto del producto como del proceso en sí, son suficientes y apropiados.

## ADQUISICIÓN DE COMPETENCIAS

Este proyecto y su documentación han hecho evidentes algunos conceptos importantes:

.- La necesidad y el coste de las pruebas en un ámbito de implementación, fase que no se suele tener en cuenta y a la que se resta importancia.

.- Inicialmente, los objetivos deben ser relativamente realistas.

.- El proceso de investigación de cualquier ciencia conlleva una inversión de tiempo superior a la aparente en un principio.

Las comparaciones después de cada sección dan los resultados que se podrían asociar con este apartado, como que heurística es superior.

# REFERENCIAS BIBLIOGRÁFICAS

Howard Phillips Lovecraft

<https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3>

<http://cs231n.github.io/convolutional-networks/>

<https://www.extremetech.com/extreme/275768-artificial-general-intelligence-is-here-and-impala-is-its-name>

<https://docs.google.com/document/d/1BgkLwn66qN0oej3QEnjjcF-4LunZpjKTA62uQlETTBw/preview?pli=1>

<https://en.wikipedia.org/wiki/Null-move_heuristic>

<https://github.com/lamesjim/Chess-AI>

<https://www.ijcai.org/Proceedings/75/Papers/048.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.563&rep=rep1&type=pdf>

<https://en.wikipedia.org/wiki/Transposition_table>

<https://stackoverflow.com/questions/41756443/how-to-implement-iterative-deepening-with-alpha-beta-pruning>

<https://www.semanticscholar.org/paper/The-History-Heuristic-and-Alpha-Beta-Search-in-Schaeffer/bb2558b0f519ea921c4aff1197555153091f7177>

<https://pdfs.semanticscholar.org/b4d2/cf76e4c42b9325b52aac45d61e80a01de77b.pdf>

<https://artint.info/html/ArtInt_62.html>

<https://stackoverflow.com/questions/753954/how-to-program-a-neural-network-for-chess>

<https://www.quora.com/How-would-somebody-model-a-neural-network-for-playing-chess>

<https://machinelearnings.co/part-1-neural-chess-player-from-data-gathering-to-data-augmentation-d51f471a61b8>

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/>

<https://stats.stackexchange.com/questions/308777/why-are-there-no-deep-reinforcement-learning-engines-for-chess-similar-to-alpha>

<https://github.com/Zeta36/chess-alpha-zero>

<https://papers.nips.cc/paper/6427-toward-deeper-understanding-of-neural-networks-the-power-of-initialization-and-a-dual-view-on-expressivity.pdf>

<https://arxiv.org/pdf/1602.05897.pdf>

<https://ai.stackexchange.com/questions/5174/what-else-can-boost-iterative-deepening-with-alpha-beta-pruning>

<https://stackoverflow.com/questions/20009796/transposition-tables>

<https://en.wikipedia.org/wiki/Zobrist_hashing>

<http://mediocrechess.blogspot.com/2007/01/guide-transposition-tables.html>

<https://www.chessprogramming.org/Refutation_Table>

<https://scholar.google.es/scholar?q=Deep+Reinforcement+Learning+keras+chess&hl=es&as_sdt=0&as_vis=1&oi=scholart>

<https://www.chessprogramming.org/>

<https://www.javiercancela.com/pymle-equations.pdf>

<https://becominghuman.ai/reinforcement-learning-step-by-step-17cde7dbc56c>

<https://ai.stackexchange.com/questions/5891/why-most-imperfect-information-games-usually-use-non-machine-learning-ai>

<https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>

<https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0>

<https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c05cfdfd3>

<https://ai.stackexchange.com/questions/7159/how-do-i-choose-which-algorithm-is-best-for-something-like-a-checkers-board-game>

<https://www.google.com/search?q=Quiescence+search&oq=Quiescence+search&aqs=chrome..69i57&sourceid=chrome&ie=UTF-8>

<https://www.intel.ai/demystifying-deep-reinforcement-learning/#gs.7ifur8>

<https://www.cs.ubc.ca/~kevinlb/teaching/cs532l%20-%202013-14/Lectures/rl-pres.pdf>

<http://stanford.edu/~jdoan21/cs221poster.pdf>

<https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/>

<https://www.baeldung.com/java-monte-carlo-tree-search>

<http://eprints.fri.uni-lj.si/1910/1/Kohne_A-1.pdf>

<https://www.google.com/search?safe=off&ei=-mPDXPfrMZSY1fAPm62_iAw&q=is+feasible+using+monte+carlo+tree+search+for+chess&oq=is+feasible+using+monte+carlo+tree+search+for+chess&gs_l=psy-ab.3..35i304i39.6410928.6412179..6412322...0.0..0.99.733.8......0....1..gws-wiz.VU4QfLh06UM>

<https://artint.info/2e/html/ArtInt2e.Ch12.S10.SS1.html>

<https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f>

<https://skymind.ai/wiki/deep-reinforcement-learning>

<https://int8.io/monte-carlo-tree-search-beginners-guide/>

<https://www.reddit.com/r/MachineLearning/comments/86s1rl/p_monte_carlo_tree_search_beginners_guide/>

<https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

<https://www.youtube.com/watch?v=-7scQpJT7uo>

<http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>

<https://www.excella.com/insights/top-3-most-popular-neural-networks>

<https://www.quora.com/What-is-the-algorithm-behind-Stockfish-the-chess-engine>

<https://es.wikipedia.org/wiki/Stockfish>

<https://www.extremetech.com/extreme/275768-artificial-general-intelligence-is-here-and-impala-is-its-name>

# ANEXO 1 – ENTORNO DE TRABAJO

# ANEXO 2 – INICIO DEL PROGRAMA