-

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](#_Toc10690086)

[ÍNDICE DE TABLAS 6](#_Toc10690087)

[ÍNDICE DE ECUACIONES 6](#_Toc10690088)

[ÍNDICE DE FIGURAS 7](#_Toc10690089)

[RESUMEN 8](#_Toc10690090)

[SUMMARY 9](#_Toc10690091)

[INTRODUCCIÓN 10](#_Toc10690092)

[PROCESO DE INVESTIGACIÓN: ESTADO DEL ARTE, IAS 12](#_Toc10690093)

[PRIMER CONTACTO 12](#_Toc10690094)

[REDES NEURONALES, EVOLUCIÓN E INCISO RAPIDO 13](#_Toc10690095)

[TIPOS DE CAPAS 14](#_Toc10690096)

[TIPOS DE REDES 15](#_Toc10690097)

[FUNCIONES DE ACTIVACION 16](#_Toc10690098)

[Aprendizaje por refuerzo, IAs generalistas 17](#_Toc10690099)

[Q-Learning 17](#_Toc10690100)

[Redes profundas Q y mejora sobre Q Learning 18](#_Toc10690101)

[TOP OF THE LINE HEURISTICAS: STOCKFISH 19](#_Toc10690102)

[Breakthrouging all the way: DEEPMIND 20](#_Toc10690103)

[ALPHAGO: ACERCAMIENTO UTILIZADO 21](#_Toc10690104)

[ENTRENAMIENTO 22](#_Toc10690105)

[ESTRUCTURA 23](#_Toc10690106)

[IMPLEMENTACIÓN Y METODOLOGÍA 25](#_Toc10690107)

[ITERACIÓN 1 – Creación de la APP base 26](#_Toc10690108)

[ITERACIÓN 2 – Creación de sistema de comunicación entre clientes. 30](#_Toc10690109)

[ITERACIÓN 3 – Investigación y creación de IA. 34](#_Toc10690110)

[DATOS FINALES DEL PROCESO ITERATIVO 38](#_Toc10690111)

[DESARROLLO/IMPLEMENTACIÓN 41](#_Toc10690112)

[LIBRERÍA MULTIMEDIA 41](#_Toc10690113)

[CLASES INTERESANTES 41](#_Toc10690114)

[LIBRERÍAS EXTERNAS 43](#_Toc10690115)

[APLICACIÓN: SAVA DROW 44](#_Toc10690116)

[Idea, definición y reglas 44](#_Toc10690117)

[Origen del juego 47](#_Toc10690118)

[ESQUEMA DE LA APLICACIÓN 49](#_Toc10690119)

[ESTADÍSTICAS DEL CÓDIGO RESULTANTE 49](#_Toc10690120)

[ESTADÍSTICAS DE LOS ARCHIVOS LOCALES RESULTANTES 49](#_Toc10690121)

[Estructuras de datos 49](#_Toc10690122)

[Algoritmos 49](#_Toc10690123)

[LUTS 49](#_Toc10690124)

[Algoritmo de control de imágenes para ahorro de memoria 49](#_Toc10690125)

[CAMINOS 49](#_Toc10690126)

[Thread Pooling 50](#_Toc10690127)

[Gráficas y datos finales 50](#_Toc10690128)

[Sistema de servidor y cliente 51](#_Toc10690129)

[ESQUEMA DEL MODELO USADO 51](#_Toc10690130)

[--- 51](#_Toc10690131)

[TABLA DE MENSAJES 51](#_Toc10690132)

[--- 51](#_Toc10690133)

[FLUJO EN UNA EJECUCIÓN 51](#_Toc10690134)

[Inteligencia Artificial 51](#_Toc10690135)

[Algoritmo de fitness 51](#_Toc10690136)

[Algoritmos de heurística 51](#_Toc10690137)

[Mejoras a estos algoritmos 51](#_Toc10690138)

[POSIBLES AMPLIACIONES 51](#_Toc10690139)

[RESULTADOS Y DISCUSIÓN 51](#_Toc10690140)

[COSTES 52](#_Toc10690141)

[CONCLUSIONES 52](#_Toc10690142)

[REFERENCIAS BIBLIOGRÁFICAS 52](#_Toc10690143)

# ÍNDICE DE TABLAS

[Tabla 1 – Evolución de las versiones de Stockfish a lo largo del tiempo 19](#_Toc10690144)

[Tabla 2 – División de horas de trabajo por iteración y por etapa. 38](#_Toc10690145)

[Tabla 3 – Número de horas individuales dedicadas a cada etapa de cada iteración 39](#_Toc10690146)

[Tabla 4 – Número de horas acumulativas dedicadas 39](#_Toc10690147)

[Tabla 5 – Porcentajes de cada etapa sobre el total de cada iteración 40](#_Toc10690148)

[Tabla 6 – Clases de **pygame** usadas en el ámbito gráfico 41](#_Toc10690149)

[Tabla 7 – Clases de **pygame** que controlan las interacciones del usuario y el resto del entorno (no gráfico) 42](#_Toc10690150)

[Tabla 8 – Clases menores de **pygame** utilizadas 42](#_Toc10690151)

[Tabla 9 – Librerías externas auxiliares 43](#_Toc10690152)

[Tabla 10 – Cifras de uso de librerías externas en la aplicación 43](#_Toc10690153)

[Tabla 11 – Tipos de personajes existentes en Sava Drow 45](#_Toc10690154)

[Tabla 12 – Elementos interactivos en Sava Drow 46](#_Toc10690155)

[Tabla 13 – Costes finales del proyecto 52](#_Toc10690156)

# ÍNDICE DE ECUACIONES

# ÍNDICE DE FIGURAS

[Ilustración 1 - Captura del juego 10](#_Toc10690157)

[Ilustración 2 – Esquema de neurona artificial 13](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690158)

[Ilustración 3 – Salida de una capa de agrupación 14](#_Toc10690159)

[Ilustración 4 – Diagrama de una convolución 14](#_Toc10690160)

[Ilustración 5 – Red neuronal con dos capas totalmente conectadas (1º y 3º) 14](#_Toc10690161)

[Ilustración 6 y 7 – Izquierda: RN Convolucional, Centro: RN Prealimentada, Derecha: RN recurrente 15](#_Toc10690162)

[Ilustración 8 – Representación gráfica de funciones de activación (Animado en Link fuente). 16](#_Toc10690163)

[Ilustración 9 – Grafo representativo del proceso de decisión de Markov 17](#_Toc10690164)

[Ilustración 10 – Izquierda: Esquema básico de una Red profunda-Q, Derecha: Arquitectura más optimizada 18](#_Toc10690165)

[Ilustración 11 – Comparación de rendimiento en un entorno con 100 acciones posibles. 18](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690166)

[Ilustración 12 – Ramificación en una partida de GO. Cada estado del juego tiene 150-250 movimientos posibles. 21](#_Toc10690167)

[Ilustración 13 – Primera etapa del entrenamiento de AlphaGo 22](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690168)

[Ilustración 14 – 2ª etapa del entrenamiento de **AlphaGo** 22](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690169)

[Ilustración 15 – 3ª etapa del entrenamiento de AlphaGo 22](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690170)

[Ilustración 16 – Estado del juego, AlphaGo 23](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690171)

[Ilustración 17 – Estructura de la red neuronal de **AlphaGo** 23](#_Toc10690172)

[Ilustración 18 – Cabecera de ajuste de valores, AlphaGo 24](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690173)

[Ilustración 19 – Cabecera de política a seguir, AlphaGo 24](file:///C:\Users\Hinjeniero\sava_drow\tfe_epcc_tfg_plantilla_memoria.docx#_Toc10690174)

[Ilustración 20 – Comparación entre esquema previo y resultado final 44](#_Toc10690175)

# 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

En el planteamiento de este trabajo de fin de grado, se divisaron los siguientes objetivos:

1.- **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.

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

3.- **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.

La idea original era tener un producto funcional, que contenga dos sistemas creados y funcionando: Comunicación entre usuarios, y agentes no humanos.

Inicialmente, el primer problema que te encuentras es que hacer de aplicación base, así como que normas a seguir tuviese, después, como hacerla para que luego soporte todas las funcionalidades que se tienen pensadas.



Ilustración - Captura del juego

Para ello, se decidió 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).

En el proceso de investigación, he indagado en la rama de **inteligencia artificial,** y adquirido conocimientos sobre los distintos tipos de inteligencias artificiales que se entienden, su aplicación y rendimiento en diversos entornos, los procesos de aprendizaje utilizados, así como el estado del arte actual, y los últimos avances.

**Explicar un poco antecedentes, y una imagen**

Tras este proceso, y no sin deliberación, se debe abandonar la idea de crear un sistema de inteligencia artificial complejo, debido a las restricciones de tiempo, dinero y medios.

Este apartado puede incluir de manera general uno o varios de los siguientes aspectos:

Intención del autor, tesis o hipótesis del trabajo.

El planteamiento del problema.

Información sobre los antecedentes.

Metas, objetivos y tipo de investigación.

# 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.

## 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.

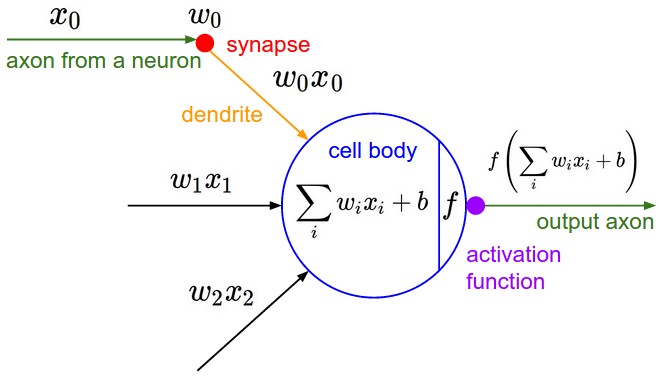
[](http://cs231n.github.io/assets/nn1/neuron_model.jpeg)Por último, cada unidad de la estructura basará su valor de disparo o activación (**triggering)**, en una función de activación decidida de antemano, y el valor de entrada en esa unidad.

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.

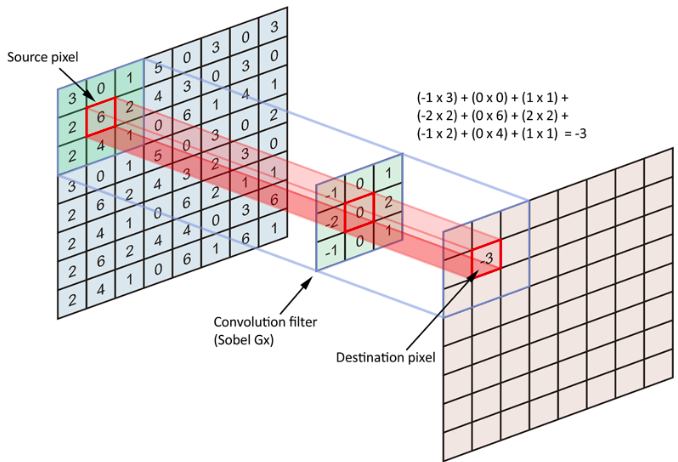
[](https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwi_p5CphcniAhUExYUKHdwPAzoQjRx6BAgBEAU&url=https%3A%2F%2Fwww.freecodecamp.org%2Fnews%2Fan-intuitive-guide-to-convolutional-neural-networks-260c2de0a050%2F&psig=AOvVaw00NeuiujYyqDq5ZMG14oSK&ust=1559504386431045)

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.

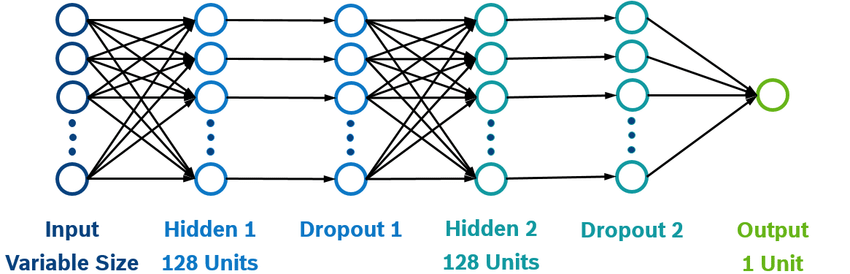
[](https://www.researchgate.net/profile/Igor_Gilitschenski/publication/311920717/figure/fig2/AS:617625239425024@1524264740078/Sample-dense-neural-network-with-2-fully-connected-layers-2-dropout-layers-and-a.png)

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.

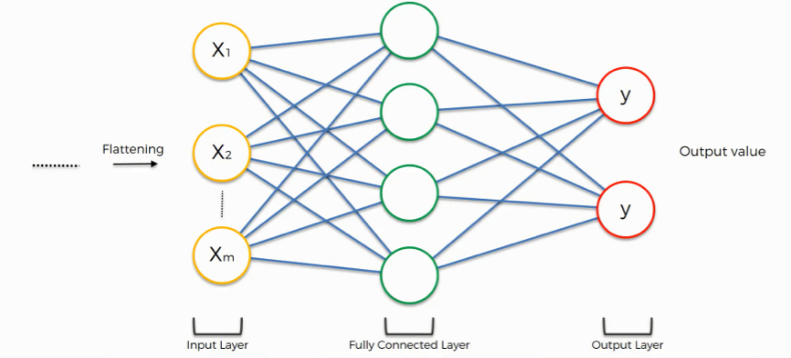
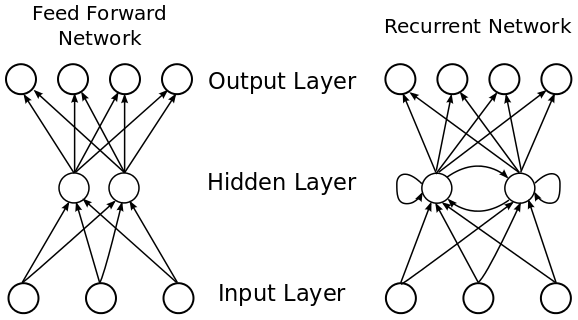
[](https://sds-platform-private.s3-us-east-2.amazonaws.com/uploads/74_blog_image_1.png)[](https://www.researchgate.net/profile/Dana_Hughes3/publication/305881131/figure/fig5/AS:391681317851147@1470395511494/Feed-forward-and-recurrent-neural-networks.png)

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).

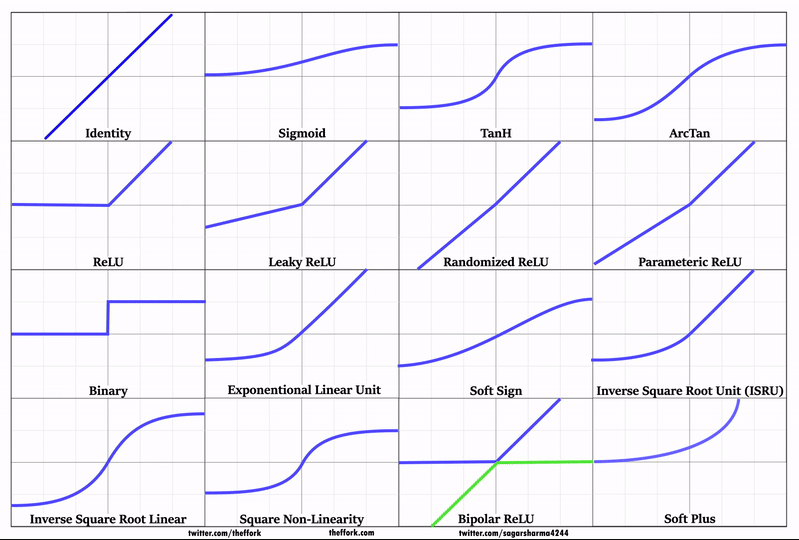
[](http://theffork.com/wp-content/uploads/2019/03/1.gif)

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**).

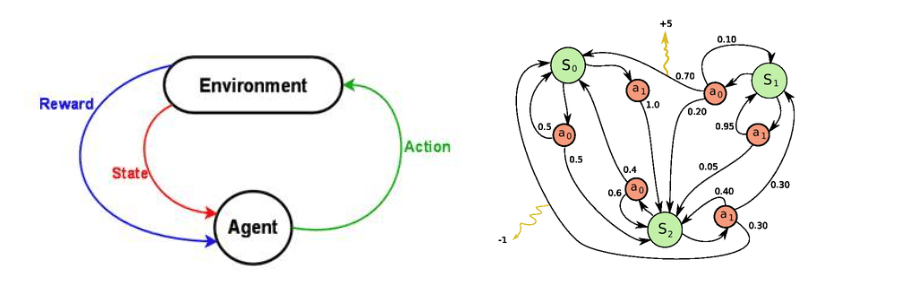
[](https://www.intel.ai/demystifying-deep-reinforcement-learning/)

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 1 – 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.

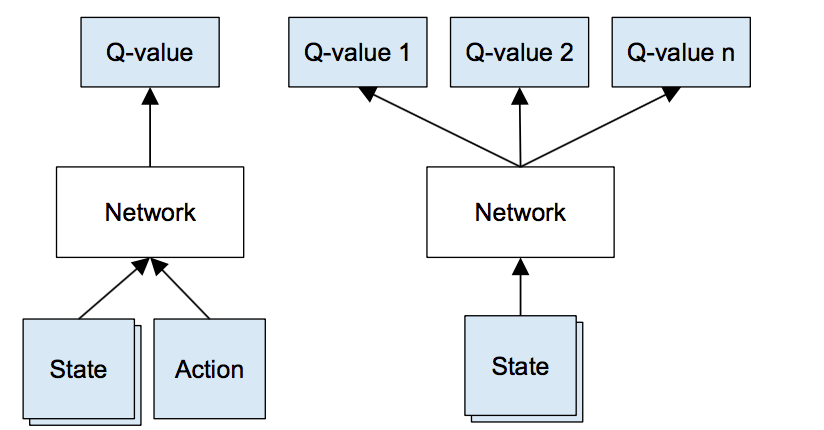
[](http://simplecore-dev.intel.com/ai/wp-content/uploads/sites/71/deep-q-network-example.png)

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**).

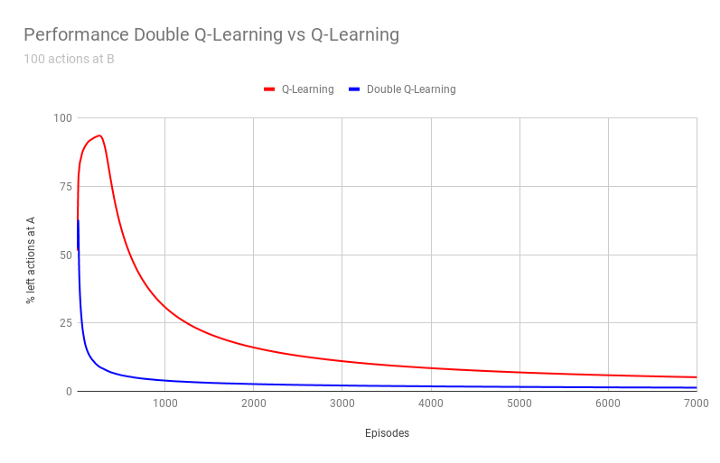
[](https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3)

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 1 – 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.

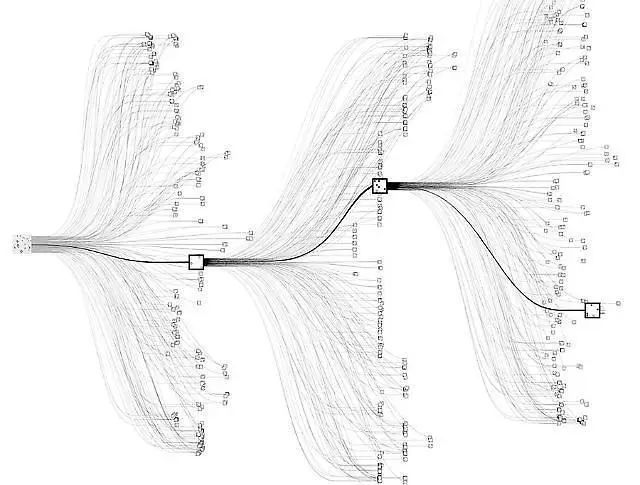
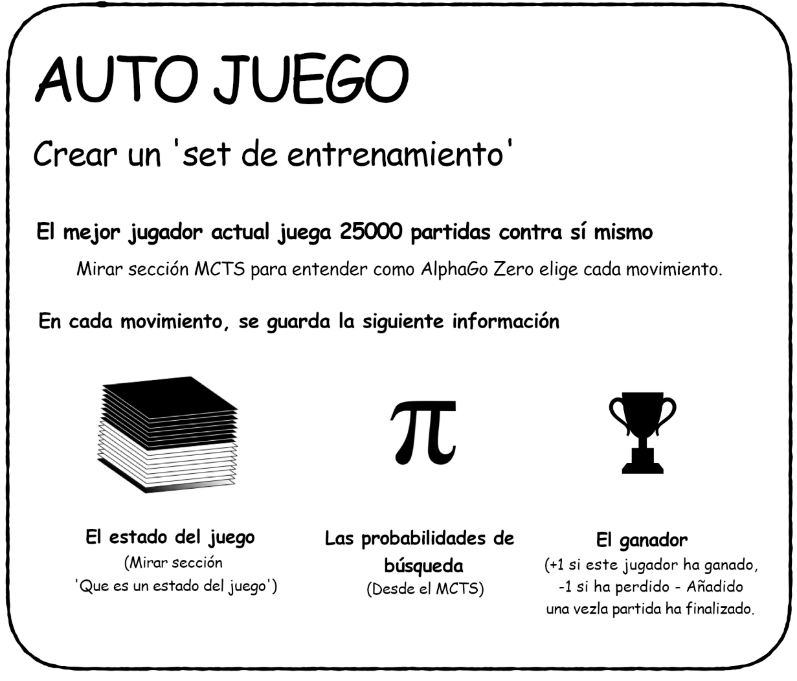


Ilustración – Ramificación en una partida de GO. Cada estado del juego tiene 150-250 movimientos posibles.

### 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.

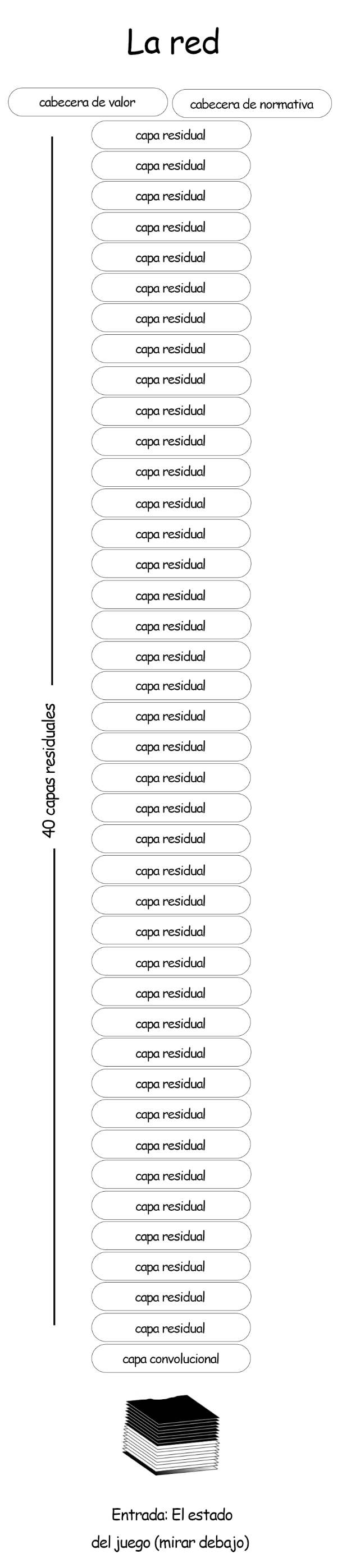
Ilustración – 2ª etapa del entrenamiento de **AlphaGo**

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



…

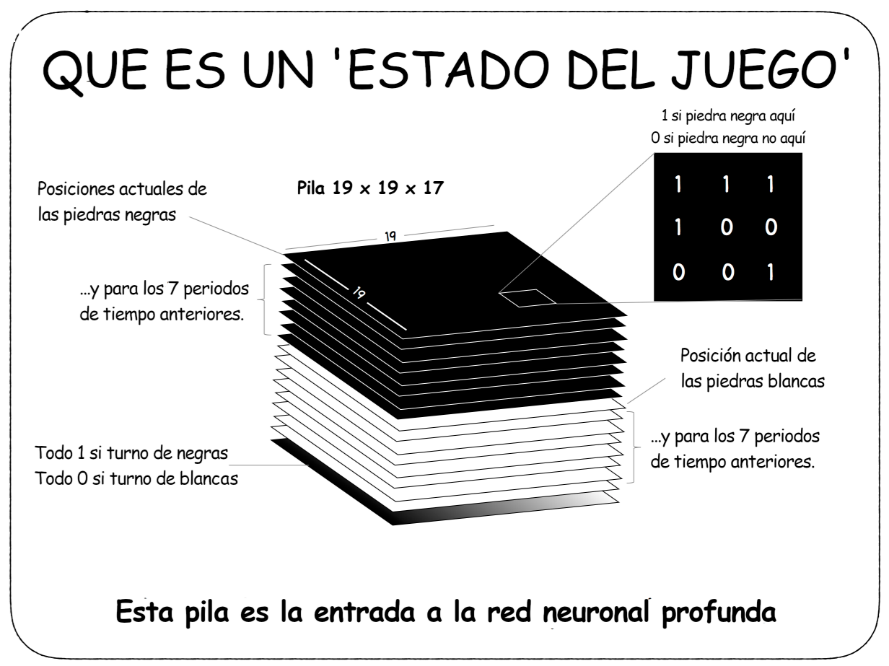
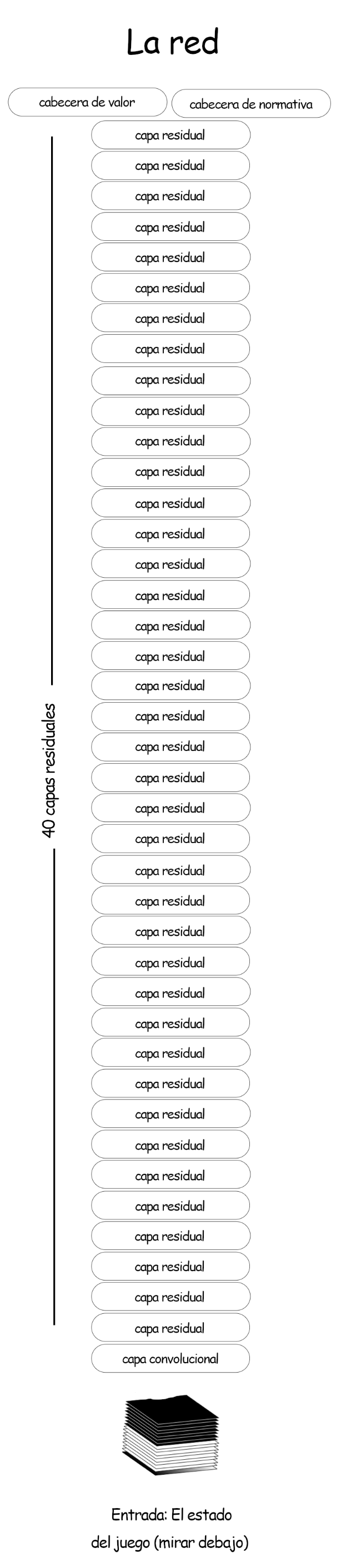


Ilustración – Estado del juego, AlphaGo



…

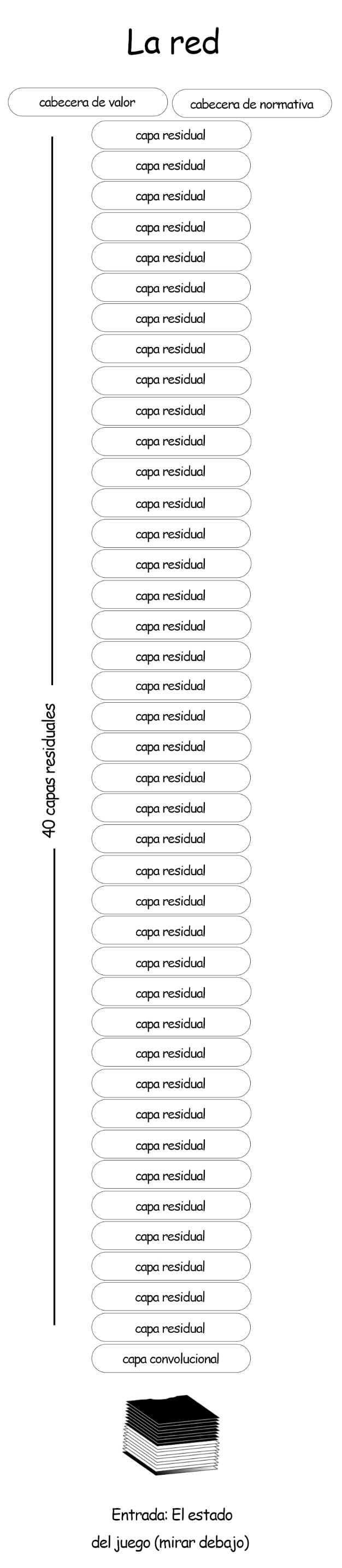


Ilustración – Estructura de la red neuronal de **AlphaGo**

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

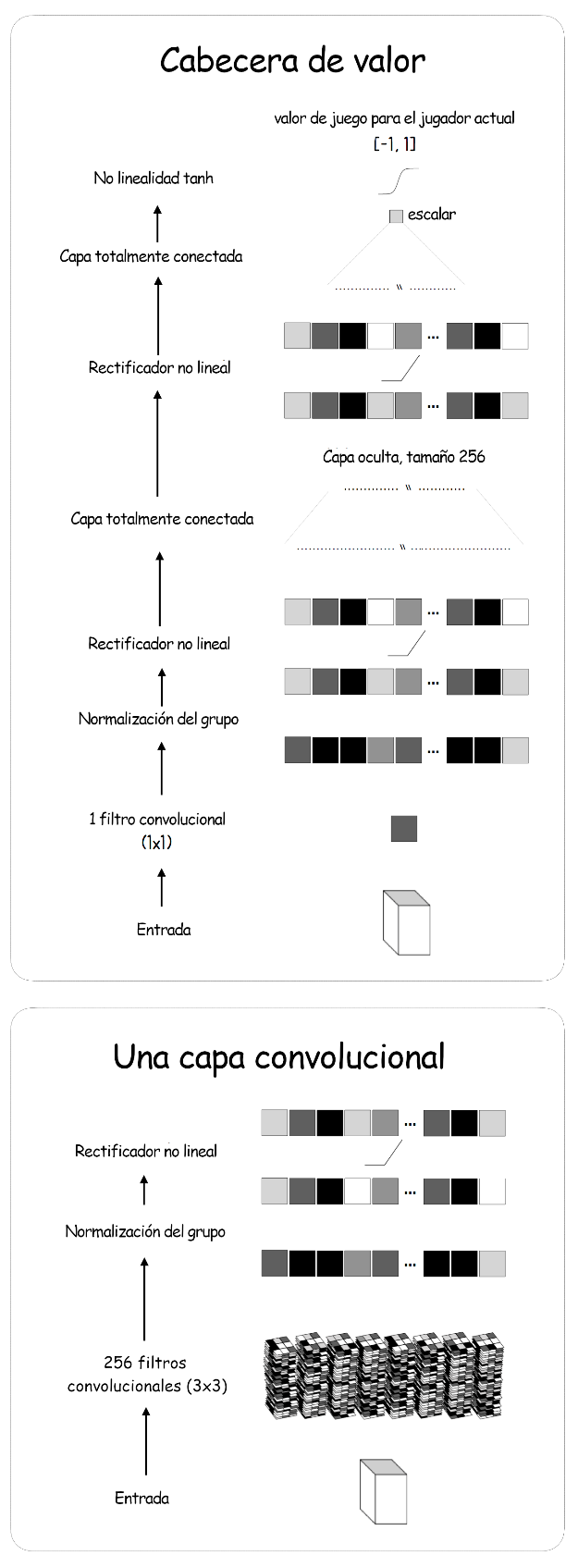
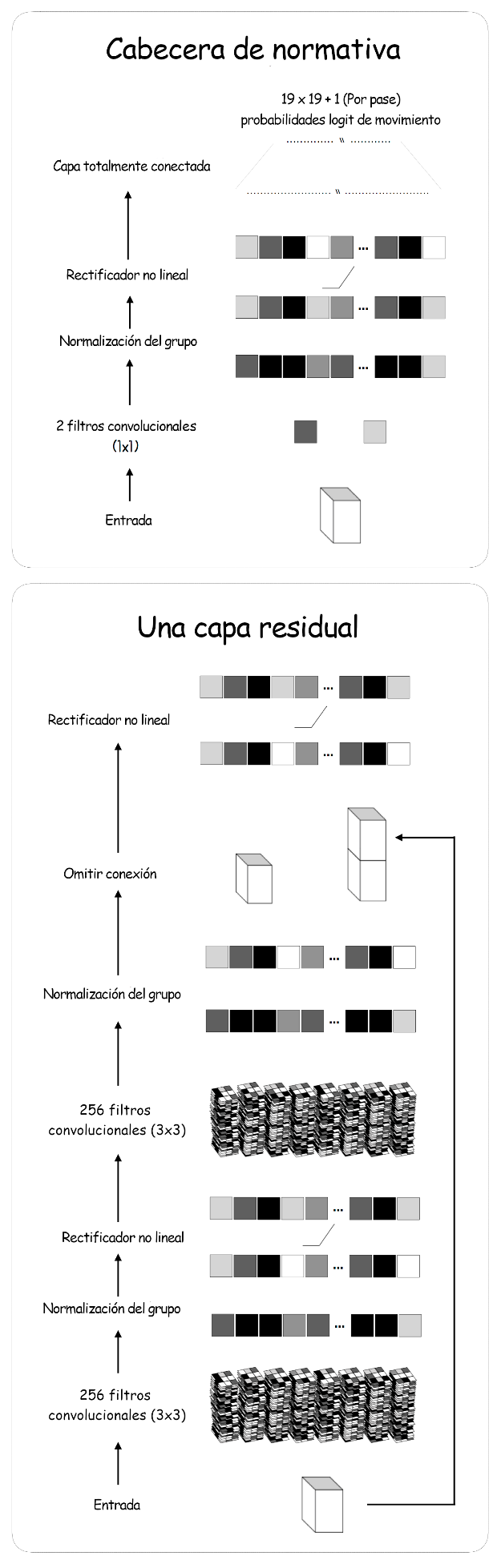


Ilustración – Cabecera de ajuste de valores, AlphaGo

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**

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.

## ITERACIÓN 1 – Creación de la 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 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, en bastantes ocasiones, el sistema operativo de los ordenadores, **Windows**, al crashear el programa por la razón que fuese, se quedaba con el puerto ocupado, por lo que para iniciar de nuevo el programa hacia falta buscar en el administrador de procesos el hilo en cuestión y matarlo 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

Se observa que, exceptuando el paso de análisis de la iteración de inteligencias artificiales (Y sólo porque incluye también mucha investigación y estudio), la mayor parte del tiempo de una ejecución completa (sumando hasta un 90%), se invierte en las etapas de implementación y pruebas, casi más en ésta última.

# 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 INTERESANTES

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.

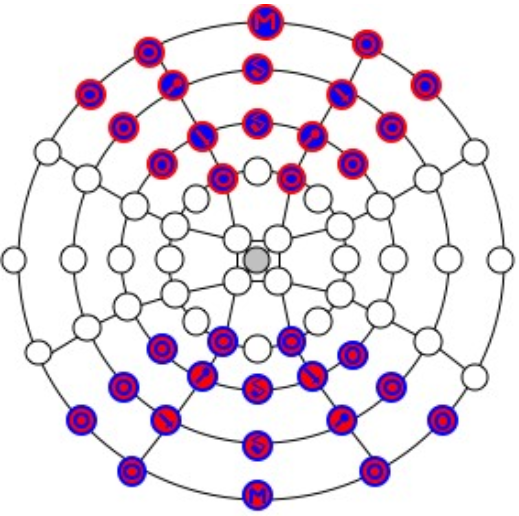
[](https://docs.google.com/document/pub?id=1BgkLwn66qN0oej3QEnjjcF-4LunZpjKTA62uQlETTBw)



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.

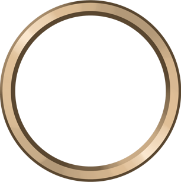


Tabla – Elementos interactivos en Sava Drow

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

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

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

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.

## 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.

TODO WAITING FO RREDDIT RESPONSE IN THE ASKING OF A PLUIGIN FRO VS CODE

### ESTADÍSTICAS DEL CÓDIGO RESULTANTE

Tabla con líneas de código por clase, ordenada de mayor a menor.

TODO

### ESTADÍSTICAS DE LOS ARCHIVOS LOCALES RESULTANTES

Tabla con tamaño de archivos locales y su tipo, ordenada de mayor a menor

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de fichero | Número de ficheros | Tamaño en disco | Tamaño acumulativo |
| Imágenes |  |  |  |
| Audio |  |  |  |
| Código python |  |  |  |
| Otros |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 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 | Tamaño=300: |  |
| Generación en tiempo real |  |  |

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.

Sin generar previamente estos cálculos, en cada movimiento de ratón se debería comprobar la distancia con los objetos que usen este método para detectar colisiones.

El objeto que usa este método es **Circle**, un tipo de Sprite cuya colisión se basa en lo anteriormente descrito.

|  |
| --- |
| 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 | Ahorro medio |
| Generación inicial |  |  |
| Generación en cada selección |  |  |

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.

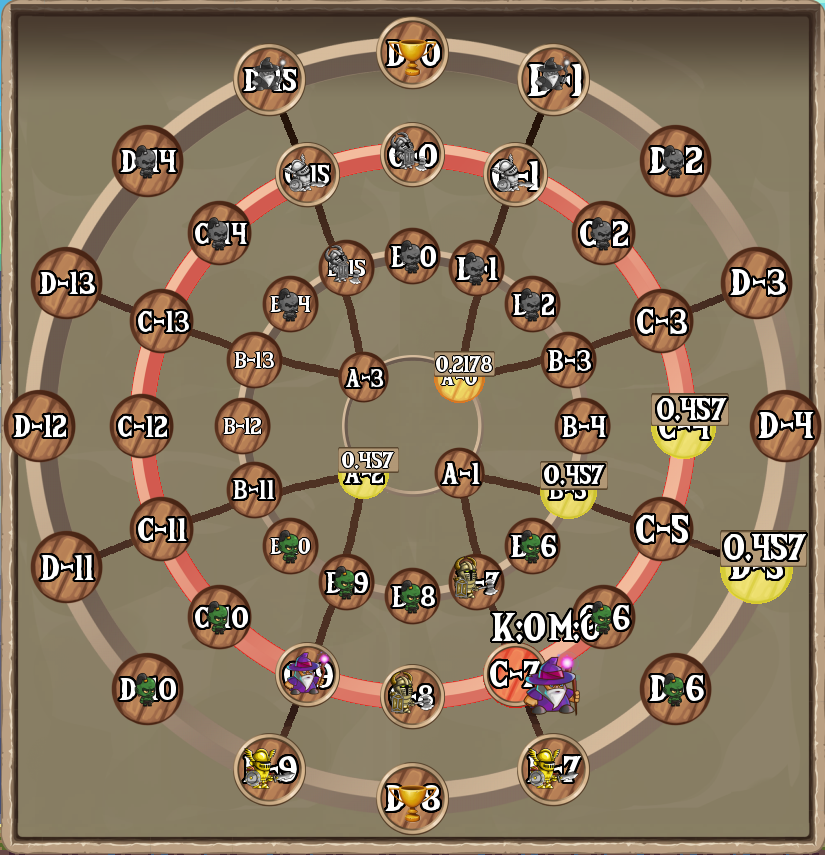


Ilustración – Ejemplo de puntuaciones ‘fitness’ en el tablero

### 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 | Tiempo de cómputo | Ahorro medio |
| Generación inicial |  |  |
| Generación en cada selección |  |  |

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

|  |  |  |
| --- | --- | --- |
| Método | Espacio en memoria | Ahorro medio |
| Usando LUT de imágenes |  |  |
| Carga normal sin LUT |  |  |

Tabla – Comparación de uso de memoria reusando imágenes mediante LUTs vs sin reusarlas.

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 un personaje

### 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.

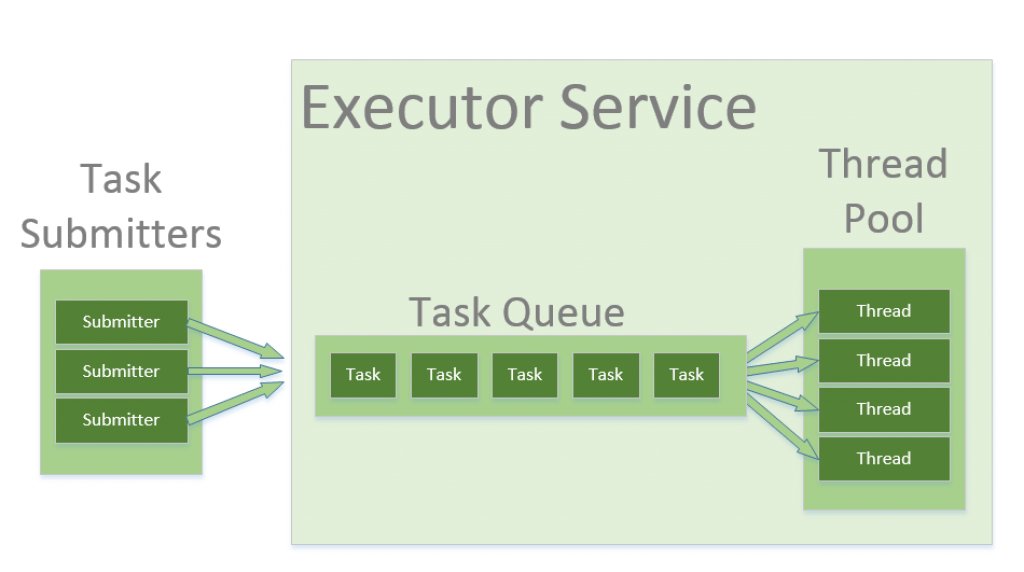


Ilustración – Representación gráfica del proceso (Agrupamiento de hilos)

### Gráficas y datos finales

#### Profiler

Profiles usado y alguna captura

#### Pruebas de rendimiento

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Equipo | Carga completa | Carga en modo test | FPS en menu | FPS en tablero |
| MSI |  |  |  |  |
| Thinkpad |  |  |  |  |
| Lenovo |  |  |  |  |
| Intel Stick |  |  |  |  |

líneas código con y sin comentarios y porcentajes sobre el total…

TODO Cuando arreglar bugs

## Sistema de servidor y cliente

### ESQUEMA DEL MODELO USADO

### ---

### TABLA DE MENSAJES

|  |  |  |  |
| --- | --- | --- | --- |
| Mensaje | Origen | Destino | Función |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### FLUJO EN UNA EJECUCIÓN

---

Inteligencia Artificial

### Algoritmo de fitness

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

1.- Puntuación de captura: Compara

2.- Puntuación de peligro:

3.- Puntuación de Cebo:

### Algoritmos de heurística

….

### Mejoras a estos algoritmos

….

# POSIBLES AMPLIACIONES

Tablas transposicionales

Redes neuronales puras o mixtas con aprendizaje con refuerzo

# RESULTADOS Y DISCUSIÓN

Muchas de estas irán en sus respectivos apartados, aquí resumen

Mejora por tablas LUT vs siempre procesando

Mejora en memoria sin control de superficies vs con control de superficies, con las resoluciones…

Comparación de algoritmos de fitness

Comparación de algoritmos heurísticos

Comparación ordenación heurísticas

## COSTES

|  |  |  |
| --- | --- | --- |
| termino | Precio | acumulativo |
| Elementos gráficos | 25$+25$ | ~44€ |
| Costes humanos | ~750hr/8€ hr | ~6044€ |
| Costes fijos (electricidad) | ~750hr/0.15kw\*0.3+  ~50hr/0.15kw\*0.04 | ~whatever |

Tabla – Costes finales del proyecto

# CONCLUSIONES

….

# 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>