



UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

<Grado en ingeniería informática en Computadores>

Trabajo Fin de Grado

<Sava Drow>



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: <---->

ÍNDICE GENERAL DE CONTENIDOS

| | |
|---|----|
| ÍNDICE GENERAL DE CONTENIDOS | 3 |
| ÍNDICE DE TABLAS | 6 |
| ÍNDICE DE ECUACIONES | 8 |
| ÍNDICE DE FIGURAS | 9 |
| RESUMEN | 11 |
| SUMMARY | 12 |
| INTRODUCCIÓN | 13 |
| PROCESO DE INVESTIGACIÓN: ESTADO DEL ARTE, IAS..... | 15 |
| PRIMER CONTACTO..... | 15 |
| REDES NEURONALES, EVOLUCIÓN E INCISO RAPIDO | 16 |
| TIPOS DE CAPAS | 17 |
| TIPOS DE REDES..... | 18 |
| FUNCIONES DE ACTIVACION..... | 19 |
| Aprendizaje por refuerzo, IAs generalistas | 20 |
| Q-Learning | 20 |
| Redes profundas Q y mejora sobre Q Learning | 21 |
| TOP OF THE LINE HEURISTICAS: STOCKFISH | 22 |
| Breakthroughing all the way: DEEPMIND | 23 |
| ALPHAGO: ACERCAMIENTO UTILIZADO..... | 24 |
| ENTRENAMIENTO..... | 25 |
| ESTRUCTURA | 26 |
| IMPLEMENTACIÓN Y METODOLOGÍA | 28 |
| ITERACIÓN 1 – Creación de la APP base | 29 |
| ITERACIÓN 2 – Creación de sistema de comunicación entre clientes..... | 33 |

| | |
|--|----|
| ITERACIÓN 3 – Investigación y creación de IA..... | 37 |
| DATOS FINALES DEL PROCESO ITERATIVO | 41 |
| DESARROLLO/IMPLEMENTACIÓN | 44 |
| LIBRERÍA MULTIMEDIA..... | 44 |
| CLASES INTERESANTES | 44 |
| LIBRERÍAS EXTERNAS..... | 46 |
| APLICACIÓN: SAVA DROW | 47 |
| Idea, definición y reglas | 47 |
| Origen e historia del juego | 50 |
| ESQUEMA DE LA APLICACIÓN | 52 |
| ESTADÍSTICAS DEL CÓDIGO RESULTANTE | 61 |
| ESTADÍSTICAS DE LOS ARCHIVOS LOCALES RESULTANTES | 61 |
| Estructuras de datos | 62 |
| ALGORITMIA..... | 64 |
| PROCESADO INICIAL, LUTS | 64 |
| CAMINOS..... | 66 |
| CONTROL DE IMÁGENES | 67 |
| THREAD POOLING | 68 |
| Gráficas y datos finales | 69 |
| Sistema de servidor y cliente | 73 |
| ESQUEMA DEL MODELO USADO..... | 73 |
| Tabla de servidores públicos (API) | 75 |
| TABLA DE MENSAJES..... | 77 |
| FLUJO EN UNA GENERACIÓN/CONEXIÓN | 78 |
| Inteligencia Artificial | 79 |
| Algoritmo de fitness | 79 |

| | |
|---|----|
| HEURÍSTICAS | 80 |
| Decisores de movimientos | 84 |
| Mejoras aplicadas | 84 |
| Resultados algoritmos..... | 85 |
| POSIBLES AMPLIACIONES..... | 85 |
| Tablas transposicionales | 85 |
| Redes neuronales, aprendizaje con refuerzo..... | 85 |
| COSTES | 86 |
| CONCLUSIONES | 87 |
| REFERENCIAS BIBLIOGRÁFICAS | 88 |

ÍNDICE DE TABLAS

| | |
|---|----|
| Tabla 1 – Evolución de las versiones de Stockfish a lo largo del tiempo | 22 |
| Tabla 2 – División de horas de trabajo por iteración y por etapa..... | 41 |
| Tabla 3 – Número de horas individuales dedicadas a cada etapa de cada iteración | 42 |
| Tabla 4 – Número de horas acumulativas dedicadas | 42 |
| Tabla 5 – Porcentajes de cada etapa sobre el total de cada iteración | 43 |
| Tabla 6 – Clases de pygame usadas en el ámbito gráfico..... | 44 |
| Tabla 7 – Clases de pygame que controlan las interacciones del usuario y el resto del entorno (no gráfico) | 45 |
| Tabla 8 – Clases menores de pygame utilizadas..... | 45 |
| Tabla 9 – Librerías externas auxiliares | 46 |
| Tabla 10 – Cifras de uso de librerías externas en la aplicación | 46 |
| Tabla 11 – Tipos de personajes existentes en Sava Drow | 48 |
| Tabla 12 – Elementos interactivos en Sava Drow | 49 |
| Tabla 13 – Clases clave del módulo gráfico | 54 |
| Tabla 14 – Clases clave del módulo de red | 55 |
| Tabla 15 - Clases clave del módulo de IA | 56 |
| Tabla 16 - Clases clave del módulo de Interacción (Elementos del tablero) | 57 |
| Tabla 17 - Clases clave del módulo de Interacción (General) | 58 |
| Tabla 18 - Clases clave del módulo de estructuras | 59 |
| Tabla 19 - Clases clave del módulo de utilidades | 60 |
| Tabla 20 – Tabla de ficheros .py, ordenados por módulo | 61 |
| Tabla 21 – Tabla de ficheros .py, ordenados por líneas de código..... | 61 |
| Tabla 22 – Tabla de tamaño conjunto de ficheros por tipo, ordenados por tamaño total | 61 |
| Tabla 23 – Estructuras de datos utilizadas y sus propósitos..... | 62 |
| Ilustración 24 – Ejemplo de cuadrantes en un tablero | 63 |
| Tabla 25 – Pseudocódigo de distancias euclídeas | 64 |
| Tabla 26 - Comparación de rendimiento usando un LUT para distancias euclídeas (Con tamaño 300*300) | 64 |

| | |
|---|----|
| Tabla 27 - Código usado para sacar el tiempo gastado en un acceso y en el cálculo de una raíz | 64 |
| Tabla 28 - Pseudocódigo de cálculo de puntuaciones de movimiento en el tablero | 65 |
| Tabla 29 - Comparación de rendimiento usando un LUT para almacenar las puntuaciones en cada turno | 65 |
| Tabla 30 - Lut inicial para los caminos de un tablero..... | 66 |
| Tabla 31 - Comparación de rendimiento usando un LUT para almacenar los caminos posibles | 66 |
| Tabla 32 – Pseudocódigo de la estructura LUT de imágenes cargadas | 67 |
| Tabla 33 – Ejemplo de uso de memoria según la técnica en uso | 67 |
| Tabla 34 – Comparación de gasto de memoria y tiempo para todas las resoluciones | 69 |
| Tabla 35 – Gasto de memoria con el tamaño de las superficies deslimitado (Resolución 1920*1080)..... | 70 |
| Tabla 36 - Ahorro de memoria y tiempo para todas las resoluciones, usando LUTs | 70 |
| Tabla 37 - Comparativa de equipos de prueba | 71 |
| Tabla 38 – Leyenda de los esquemas de red | 73 |
| Tabla 39 – Ejemplo de dirección DNS de máquina virtual de Amazon..... | 76 |
| Tabla 40 – Mensajes usados en la conexión y sincronización de usuarios..... | 77 |
| Tabla 41 – Mensajes utilizados en el transcurso de una sesión de juego | 77 |
| Tabla 42 – Proceso de generación de una partida en un entorno de red (Clientes y Servidor)..... | 78 |
| Tabla 43 – Pseudocódigo del algoritmo de poda alfa-beta utilizado | 81 |
| Tabla 44 – Pseudocódigo del algoritmo de búsqueda MonteCarlo utilizado | 83 |
| Tabla 45 – Tabla de generadores de movimientos en tablero no heurísticos | 84 |
| Tabla 46 – Cambios realizados en el pseudocódigo de alfa beta al incluir ordenación | 84 |
| Tabla 47 – Comparación de decisores de movimiento..... | 85 |
| Tabla 48 – Comparación de rendimiento de heurísticas según tiempo | 85 |
| Tabla 49 – Costes finales del proyecto..... | 86 |

ÍNDICE DE ECUACIONES

| | |
|---|----|
| Ecuación 1 – Fórmula de Bellman | 21 |
| Ecuación 2 – Fórmula del multiplicador de peligro | 79 |
| Ecuación 3 - Fórmula de la puntuación de peligro | 79 |
| Ecuación 4 – Fórmula de la puntuación de captura..... | 79 |
| Ecuación 5 – Fórmula de la puntuación de cebo | 79 |
| Ecuación 6 – Fórmula final del algoritmo de fitness | 80 |
| Ecuación 7 – Fórmula del límite de confianza superior aplicado a árboles usado en MonteCarlo | 82 |

ÍNDICE DE FIGURAS

| | |
|--|----|
| Ilustración 1 - Captura del juego | 13 |
| Ilustración 2 – Esquema de neurona artificial | 16 |
| Ilustración 3 – Salida de una capa de agrupación..... | 17 |
| Ilustración 4 – Diagrama de una convolución..... | 17 |
| Ilustración 5 – Red neuronal con dos capas totalmente conectadas (1º y 3º)..... | 17 |
| Ilustración 6 y 7 – Izquierda: RN Convolutinal, Centro: RN Prealimentada, Derecha: RN recurrente..... | 18 |
| Ilustración 8 – Representación gráfica de funciones de activación (Animado en Link fuente)..... | 19 |
| Ilustración 9 – Grafo representativo del proceso de decisión de Markov | 20 |
| Ilustración 10 – Izquierda: Esquema básico de una Red profunda-Q, Derecha: Arquitectura más optimizada..... | 21 |
| Ilustración 11 – Comparación de rendimiento en un entorno con 100 acciones posibles. | 21 |
| Ilustración 12 – Ramificación en una partida de GO. Cada estado del juego tiene 150-250 movimientos posibles. | 24 |
| Ilustración 13 – Primera etapa del entrenamiento de AlphaGo..... | 25 |
| Ilustración 14 – 2ª etapa del entrenamiento de AlphaGo | 25 |
| Ilustración 15 – 3ª etapa del entrenamiento de AlphaGo | 25 |
| Ilustración 16 – Estado del juego, AlphaGo | 26 |
| Ilustración 17 – Estructura de la red neuronal de AlphaGo | 26 |
| Ilustración 18 – Cabecera de ajuste de valores, AlphaGo | 27 |
| Ilustración 19 – Cabecera de política a seguir, AlphaGo | 27 |
| Ilustración 20 – Iteración de un proceso iterativo e incremental | 28 |
| Ilustración 21 – Esquema completo del proceso iterativo | 28 |
| Ilustración 22 – Comparación entre esquema previo y resultado final..... | 47 |
| Ilustración 23 – Esquema simplificado de la estructura de la aplicación | 52 |
| Ilustración 24 – Diagrama UML del módulo gráfico (Excepto UIElement) | 53 |
| Ilustración 25 - Diagrama UML del módulo gráfico (Solo UIElement)..... | 53 |
| Ilustración 26 – Diagrama UML del módulo de red (Python) | 55 |

| | |
|--|----|
| Ilustración 27 – Diagrama UML del módulo de red (JavaScript) | 55 |
| Ilustración 28 - Diagrama UML del módulo de IA..... | 56 |
| Ilustración 29 - Diagrama UML del módulo de interacción | 57 |
| Ilustración 30 - Diagrama UML del módulo de estructuras..... | 59 |
| Ilustración 31 - Diagrama UML del módulo de utilidades | 60 |
| Ilustración 32 – Ejemplo de puntuaciones ‘fitness’ en el tablero..... | 65 |
| Ilustración 33 – Atributos de un objeto Restricción de un personaje | 66 |
| Ilustración 34 - Atributos de un objeto ResizedSurface de una imagen..... | 67 |
| Ilustración 35 - Flujo de paso de las tareas mediante Thread Pooling | 68 |
| Ilustración 36 – Representación gráfica del proceso (Agrupamiento de hilos)..... | 68 |
| Ilustración 37 – Esquema de comunicación en una conexión a un servidor privado | 74 |
| Ilustración 38 – Esquema de comunicación en una conexión a un servidor privado | 74 |
| Ilustración 39 – Todos los puntos de acceso del servidor de Express | 75 |
| Ilustración 40 – Panel de ajustes de Noip | 76 |
| Ilustración 41 – Panel de control de instancias en el servicio web de Amazon (AWS) | 76 |
| Ilustración 42 – Ejemplo de ejecución MiniMax..... | 80 |
| Ilustración 43 – Representación gráfica del árbol resultante tras una ejecución de poda alfa beta | 81 |
| Ilustración 44 – Representación gráfica proceso del MCTS..... | 83 |

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 possibly 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 transponen 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 1 - 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.

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.

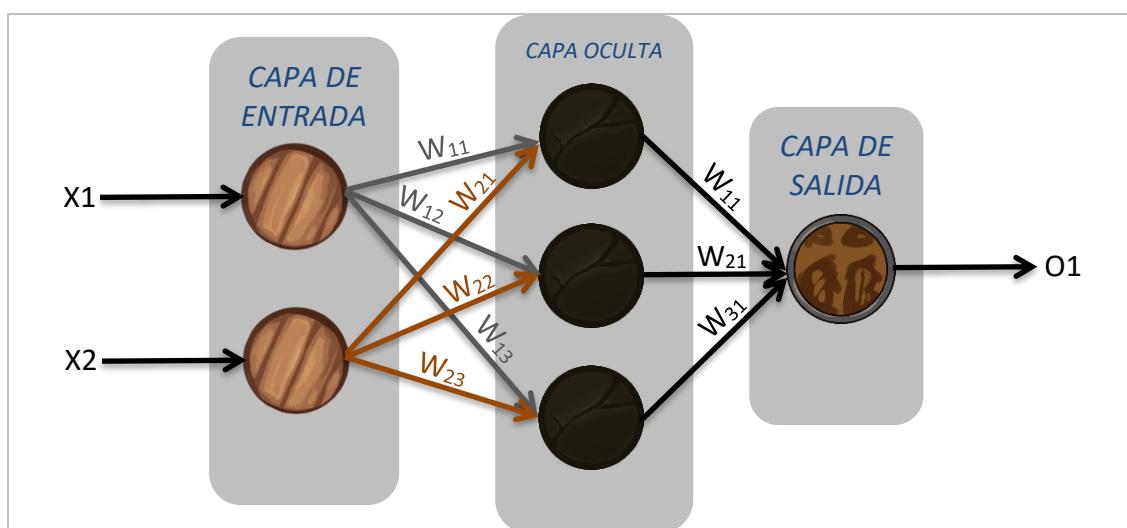


Ilustración 2 – Perceptron Multicapa

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.

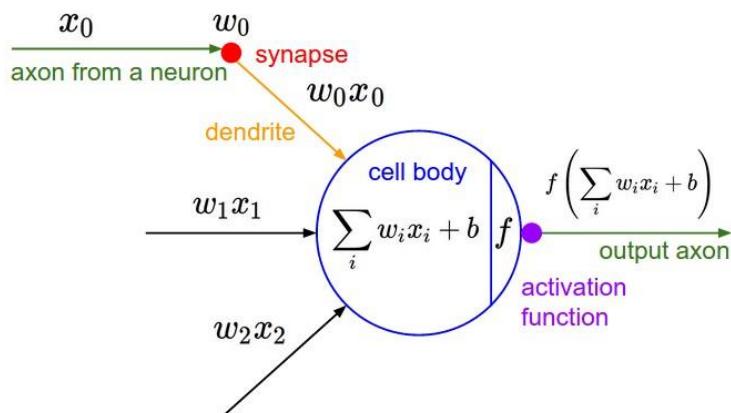
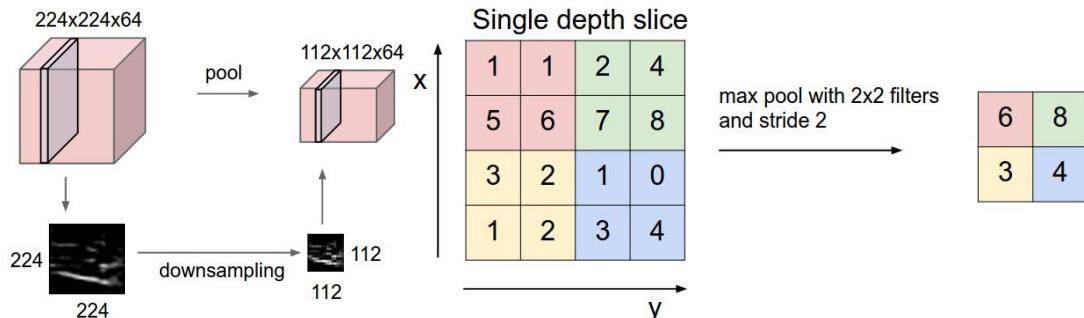


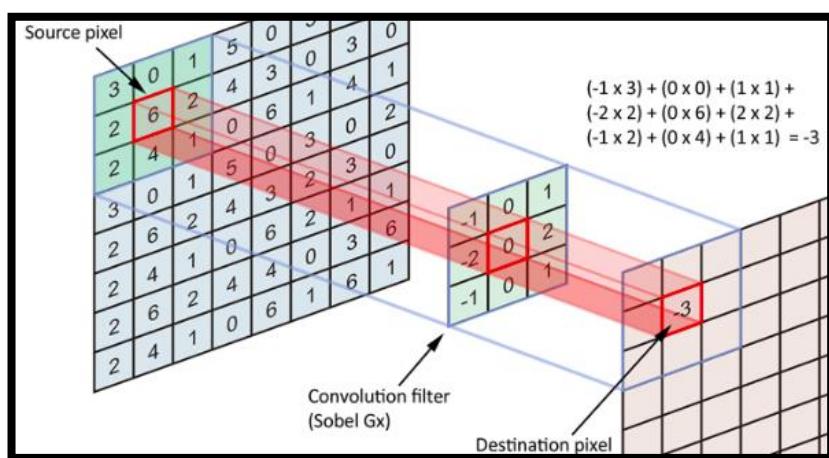
Ilustración 2 – Esquema de neurona artificial

TIPOS DE CAPAS

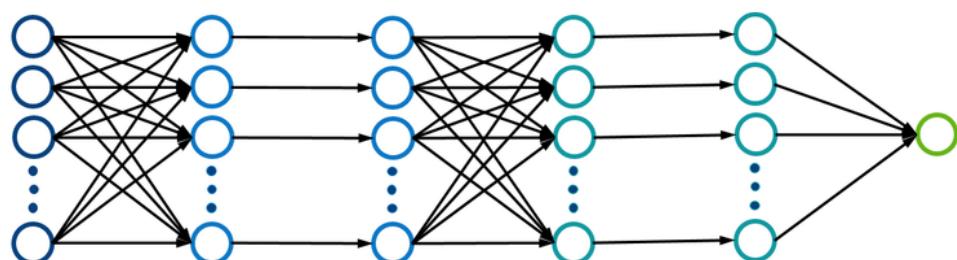
Capas de agrupación (Pooling), que permiten que la red neuronal reconozca patrones ya vistos, aunque la entrada esté en una posición distinta al ejemplo de entrenamiento.



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.



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



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 Convolutional (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.

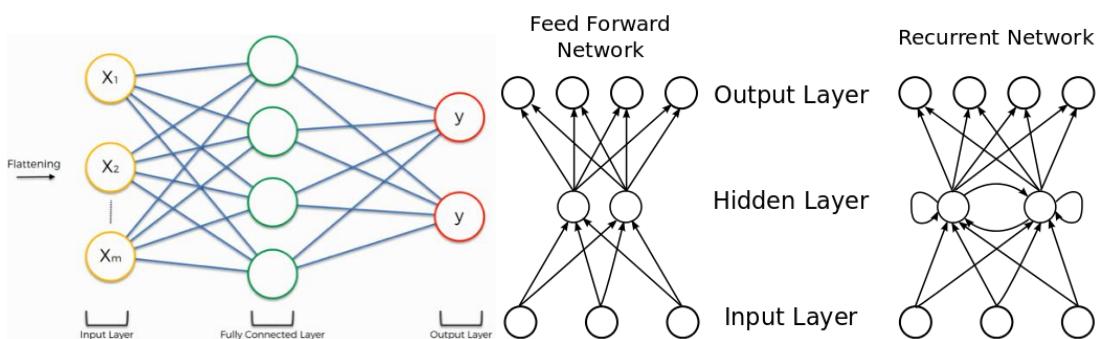


Ilustración 6 y 7 – Izquierda: RN Convolutinal, Centro: RN Prealimentada, Derecha: RN recurrente

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).

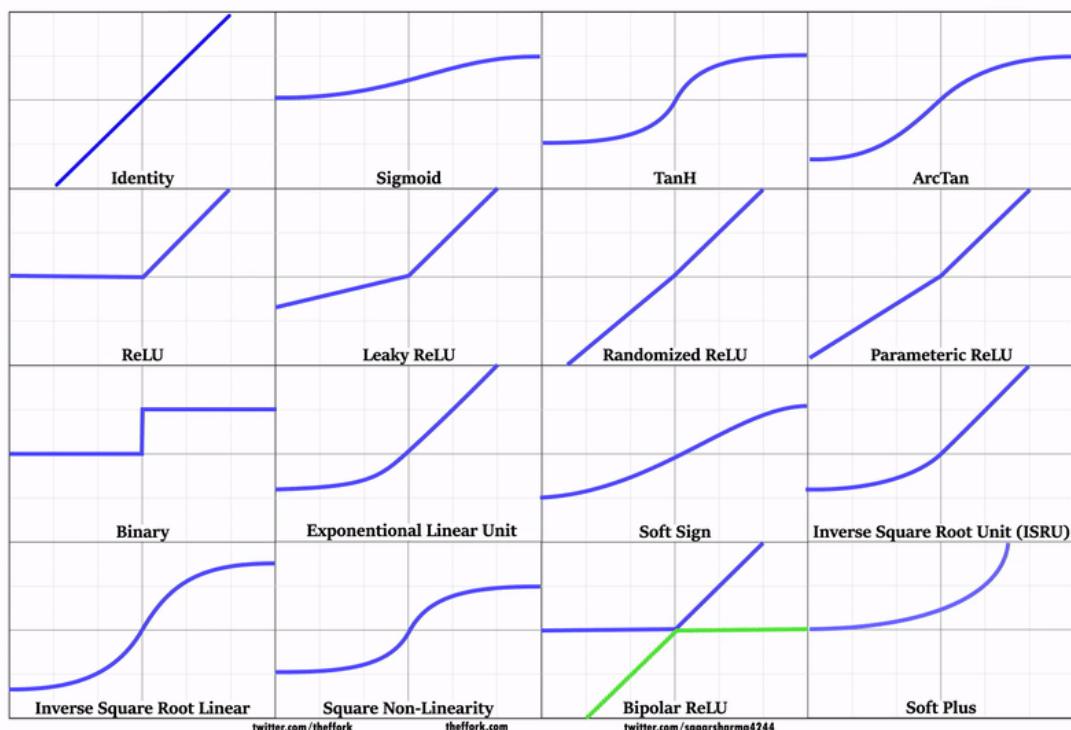


Ilustración 8 – Representación gráfica de funciones de activación (Animado en Link fuente).

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**).

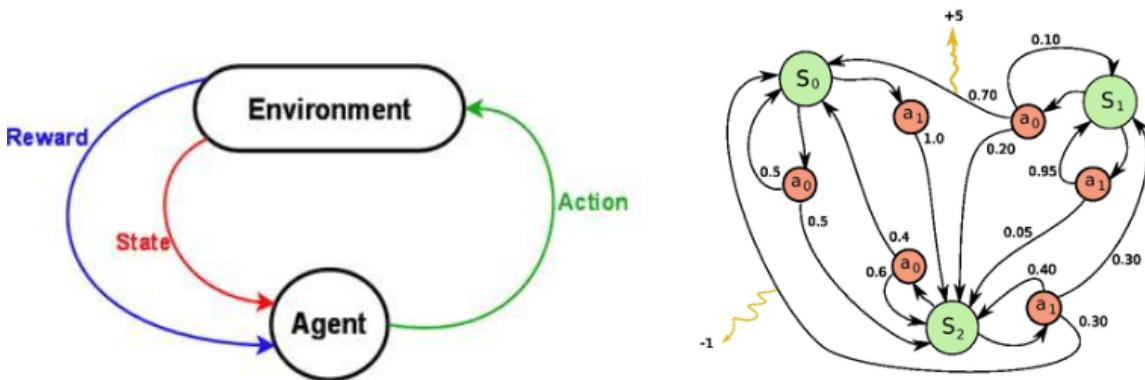


Ilustración 9 – Grafo representativo del proceso de decisión de Markov

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.

$$Q(s, a) = r + \gamma * \max a' * Q(s', a')$$

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.

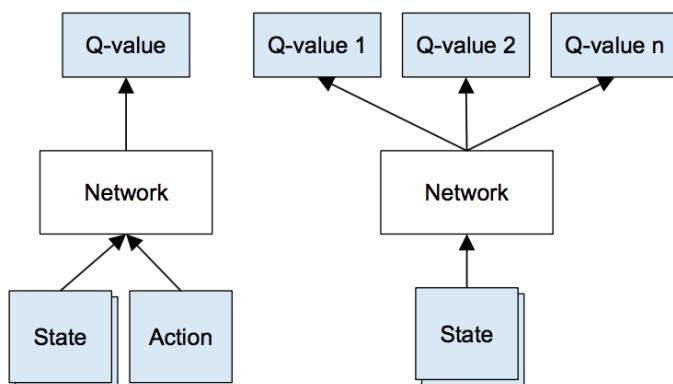


Ilustración 10 – Izquierda: Esquema básico de una Red profunda-Q, Derecha: Arquitectura más optimizada

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**).

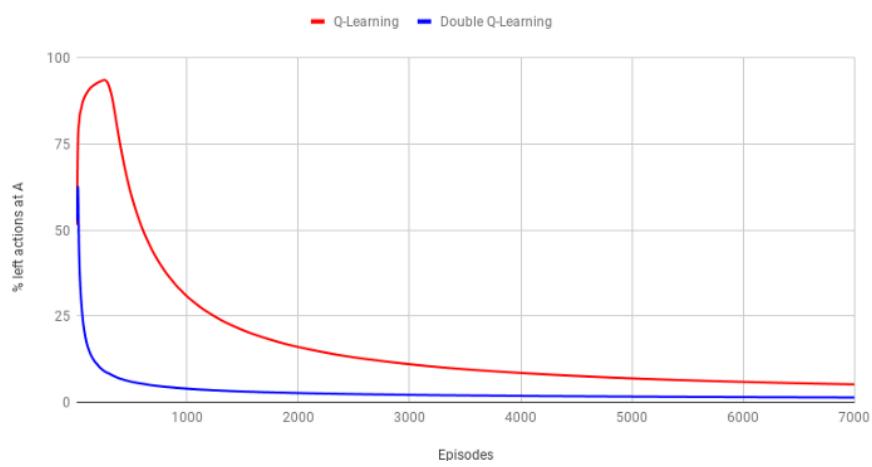


Ilustración 11 – Comparación de rendimiento en un entorno con 100 acciones posibles.

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>

Breakthrouguing 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* entrena 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=MvFABFWPBhw>

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.

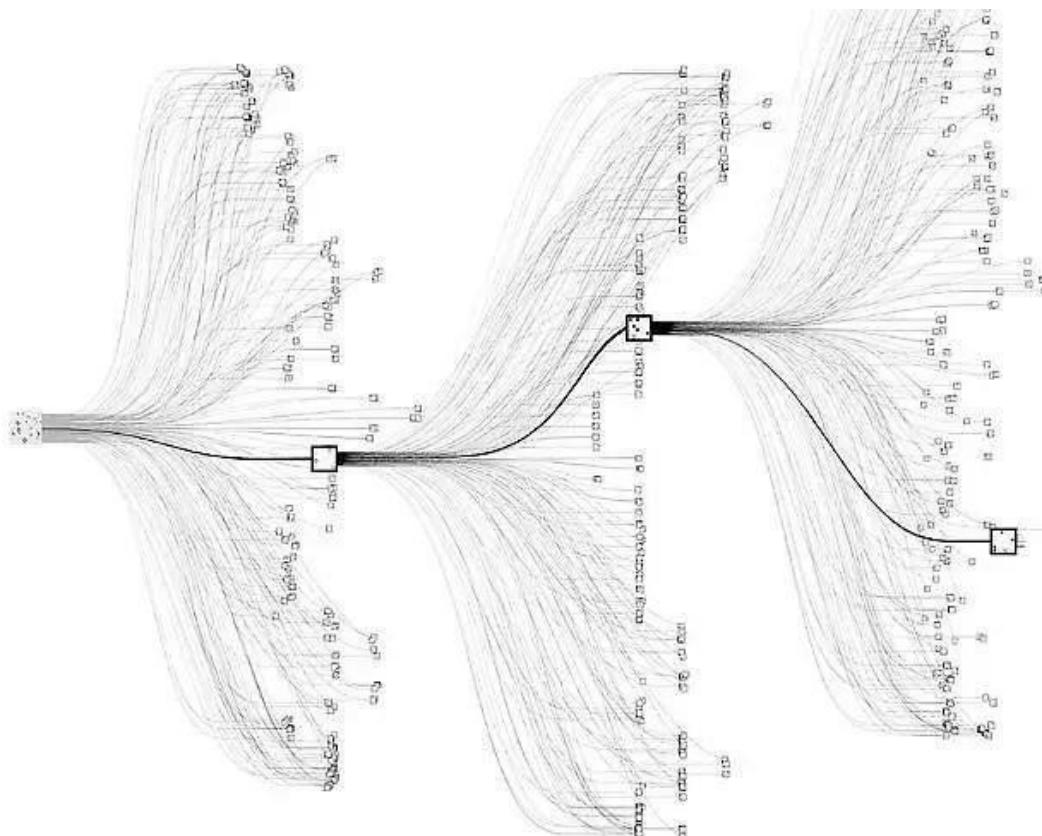


Ilustración 12 – Ramificación en una partida de GO. Cada estado del juego tiene 150-250 movimientos posibles.

ENTRENAMIENTO

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

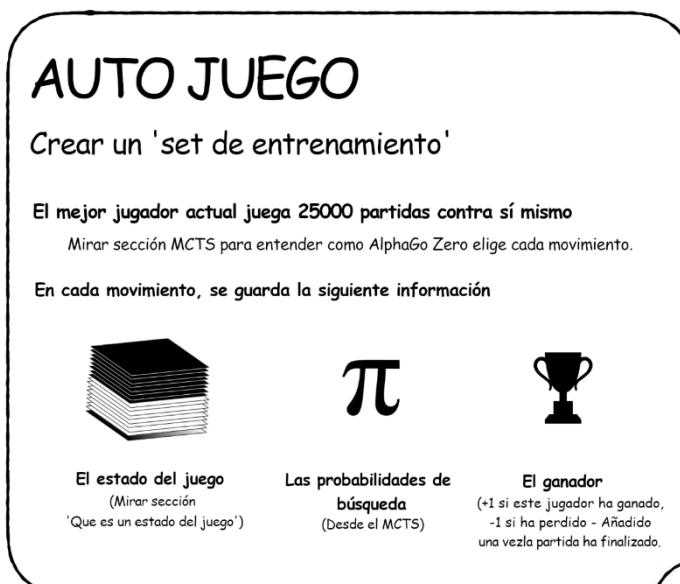


Ilustración 13 – Primera etapa del entrenamiento de AlphaGo

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.

EVALUAR RED

Test para ver si la nueva red es mejor

Ejecuta 400 partidas entre la última red neuronal y la mejor red neuronal actual

Ambos jugadores usan MCTS para elegir sus movimientos, usando sus respectivas redes neuronales para evaluar nodos hoja

El jugador más reciente debe ganar el 55% de partidas para ser declarado el nuevo mejor jugador

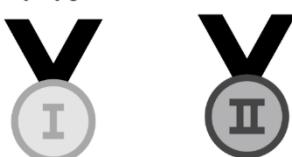


Ilustración 15 – 3^a etapa del entrenamiento de AlphaGo

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.

REENTRENAR RED

Optimizar los pesos de la red

UNA ITERACIÓN EN EL ENTRENAMIENTO

Coger una muestra de 2048 posiciones de las últimas 500,000 partidas

Reentrenar la red neuronal actual en esas posiciones

- Los estados del juego son la entrada (Mirar 'Arquitectura de Red Neuronal Profunda')

Función de pérdida

Compara predicciones de la red neuronal con las probabilidades de búsqueda y el ganador real.



Tras 1,000 iteraciones de entrenamiento, se evalúa la red

Ilustración 14 – 2^a 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.

ESTRUCTURA

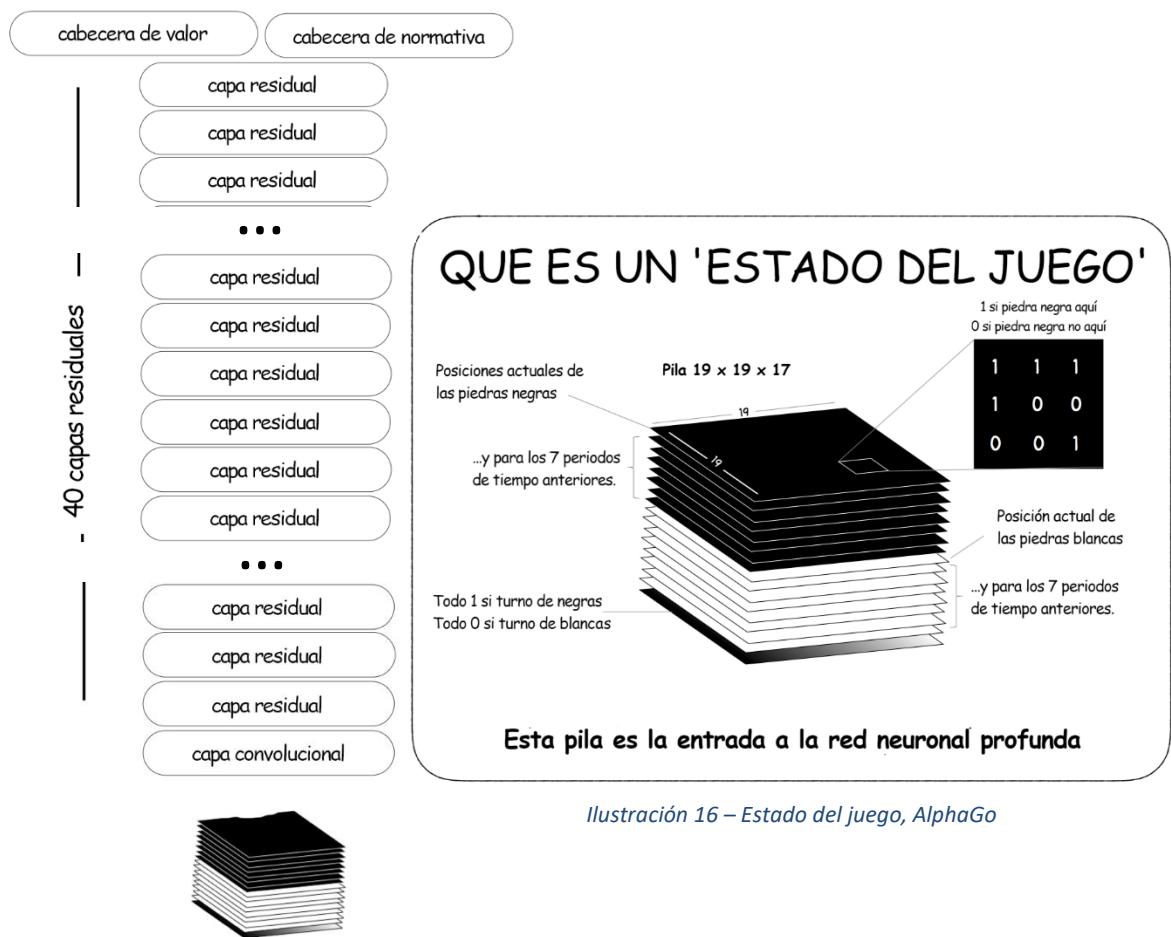


Ilustración 17 – Estructura de la red neuronal de AlphaGo

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).

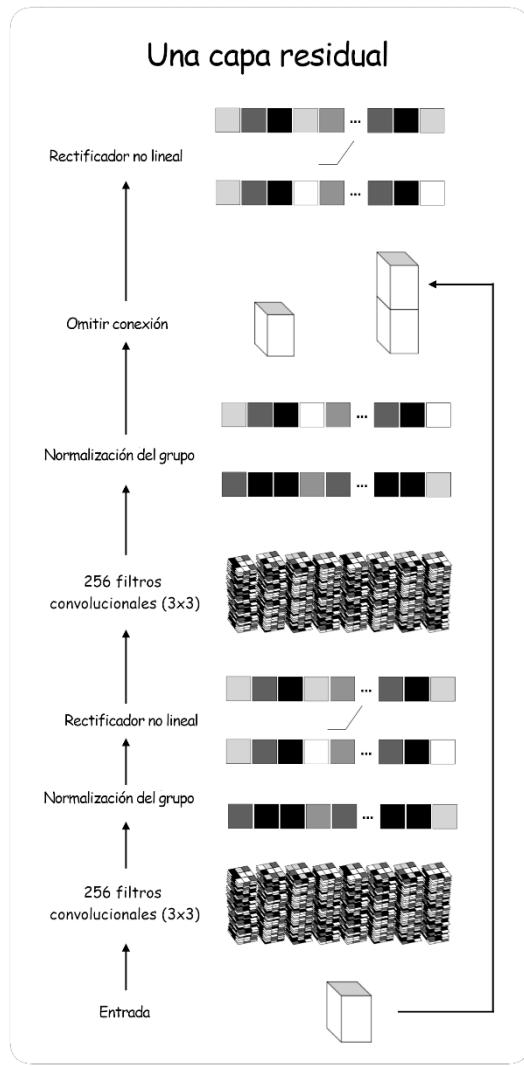
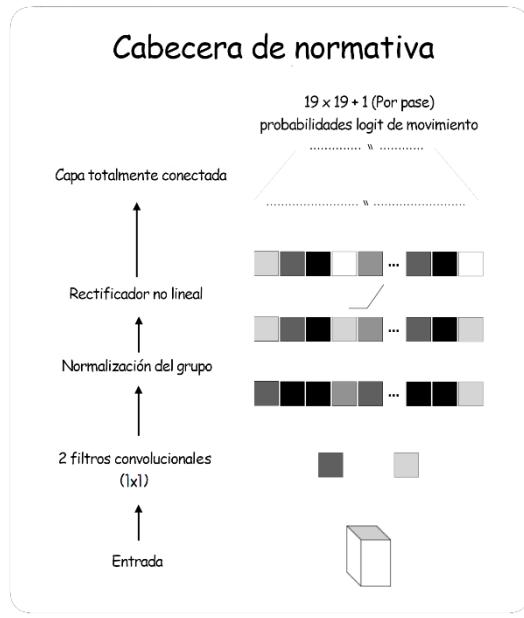
Las capas convolucionales reducen el tamaño de los datos mediante filtros, y las 40 capas residuales

Las dos cabeceras finales normalizan y ajustan la salida, en función de la política que se esté siguiendo en ese momento.

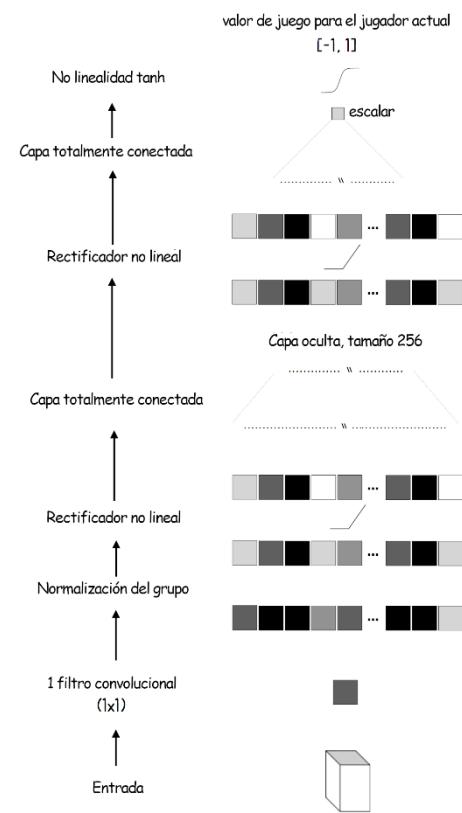
Sus contenidos se muestran gráficamente a continuación.

Servirán para cerrar el capítulo que explica los interiores de AlphaGo

Ilustración 19 – Cabecera de política a seguir, AlphaGo



Cabecera de valor



Una capa convolucional

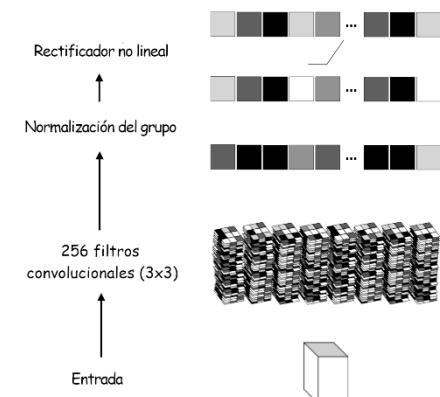


Ilustración 18 – Cabecera de ajuste de valores, AlphaGo

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).

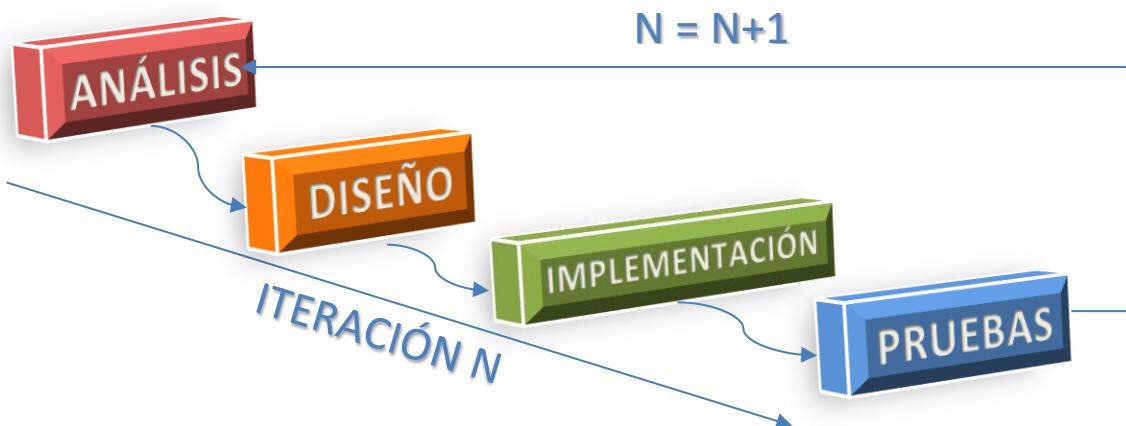


Ilustración 20 – Iteración de un proceso iterativo e incremental

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.

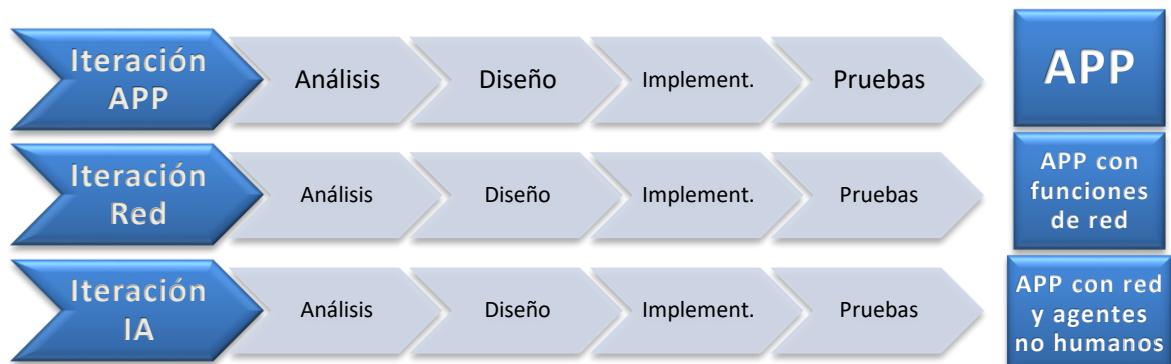


Ilustración 21 – Esquema completo del proceso iterativo

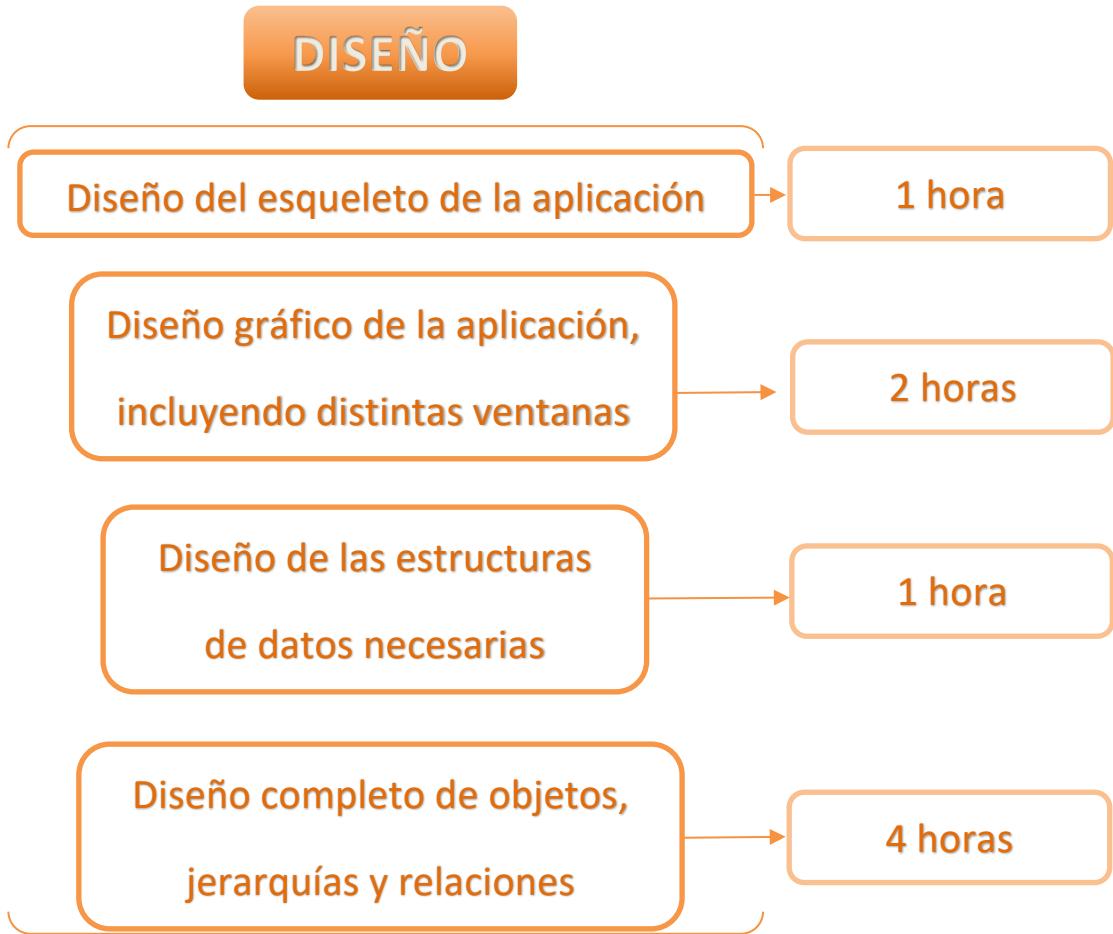
ITERACIÓN 1 – Creación de la APP base



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.

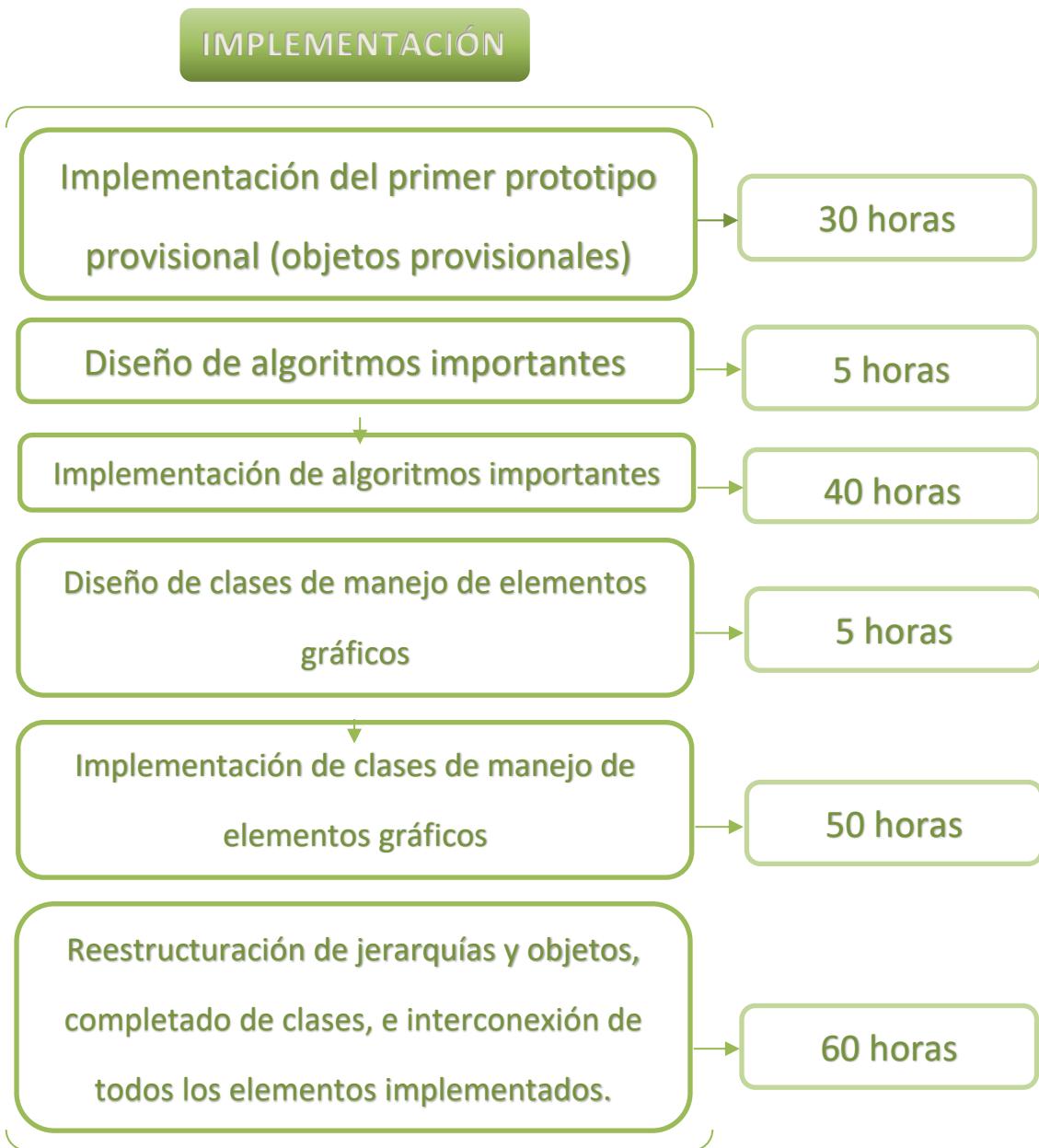
Como no importa si la aplicación gráfica es en 2 dimensiones en lugar de 3, y saliendo su última versión poco antes de la época en la que tuvo lugar este paso, se eligió **pygame**. En su análisis se estudió la documentación de la librería.



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.



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

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

Test y pruebas de algoritmos de manejo de superficies y archivos

10 horas

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.



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

Esquema de mensajes necesarios

2 horas

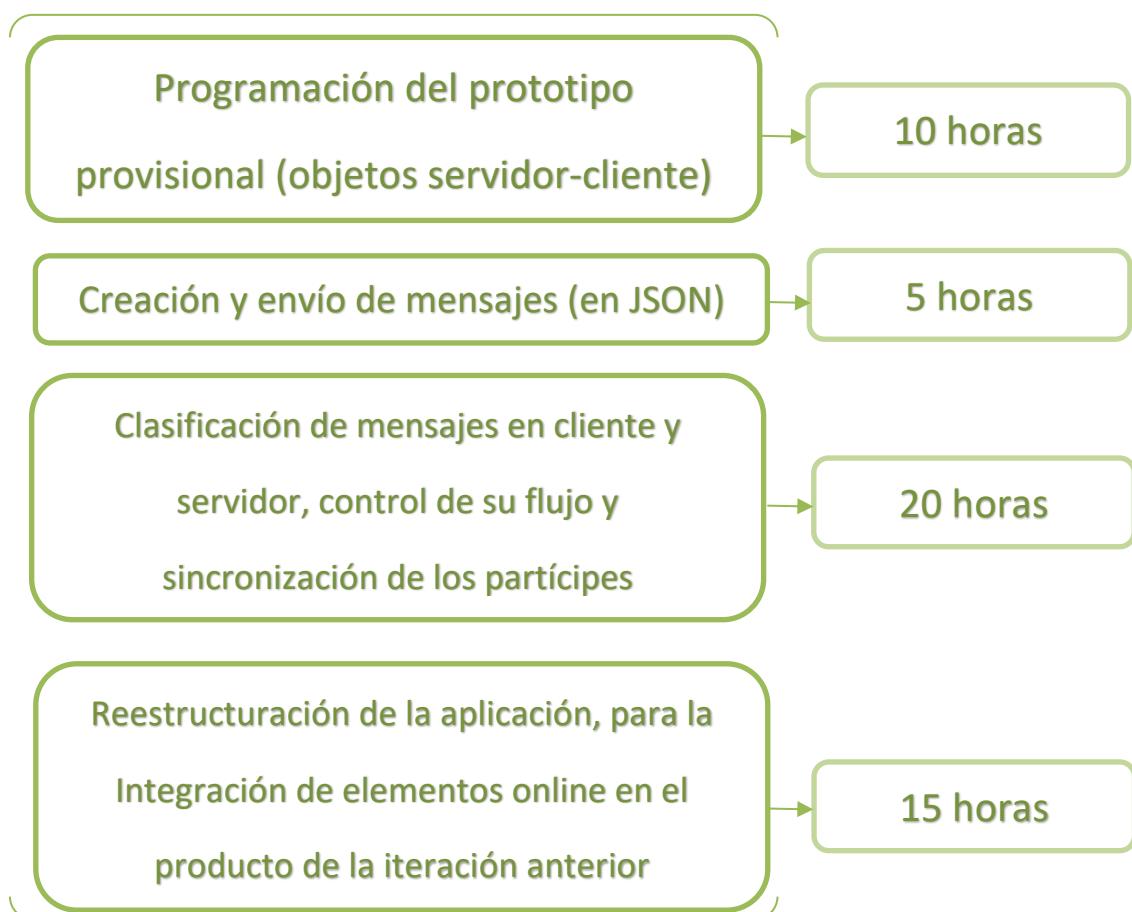
Diagrama de flujo de mensajes y su intercambio

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 ir en la codificación en sí.

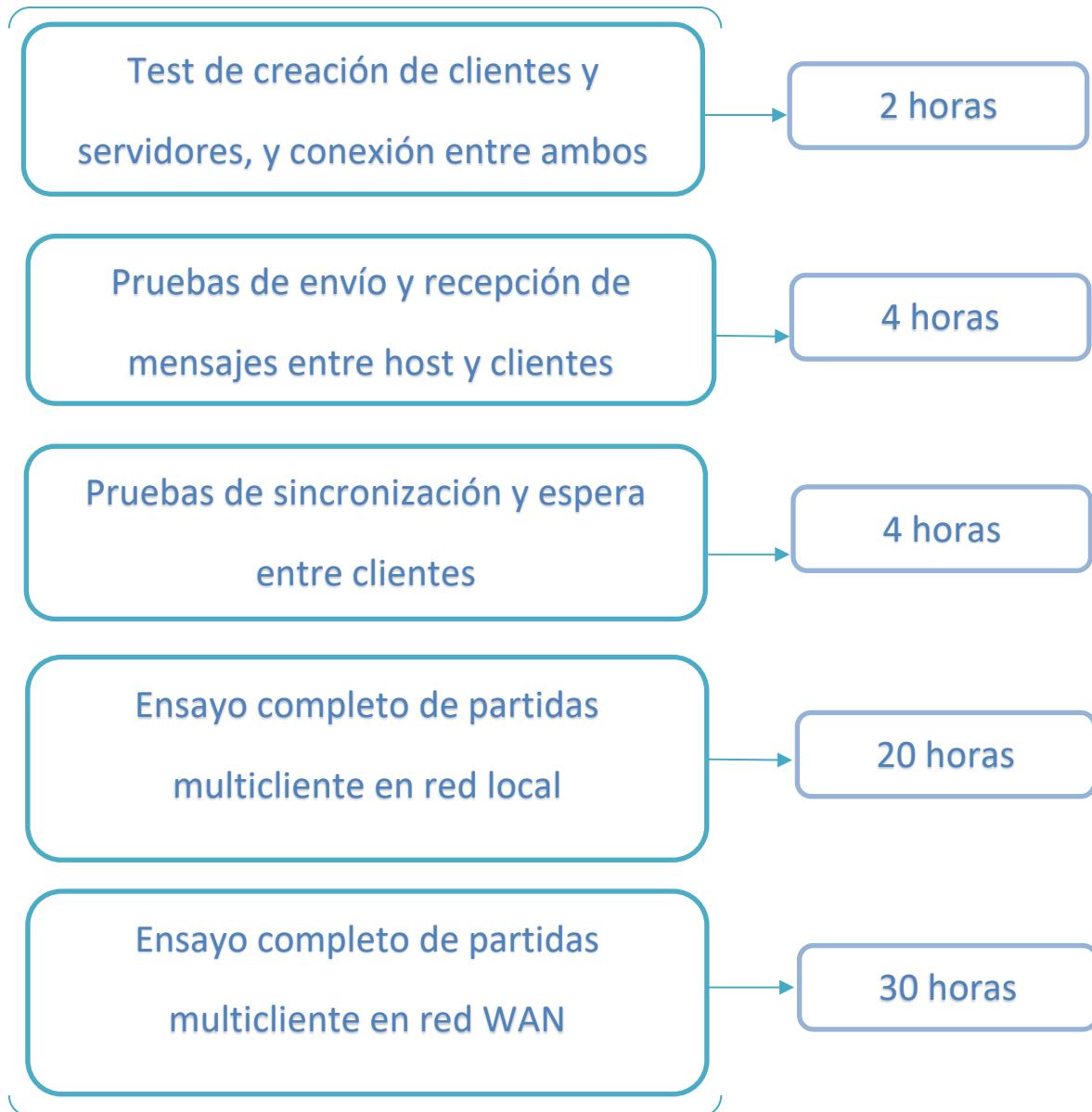
IMPLEMENTACIÓN



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.

PRUEBAS



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

Investigación del panorama actual en Inteligencia artificial → 10 horas

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 caso 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.



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 manejar las estructuras de los algoritmos anteriores

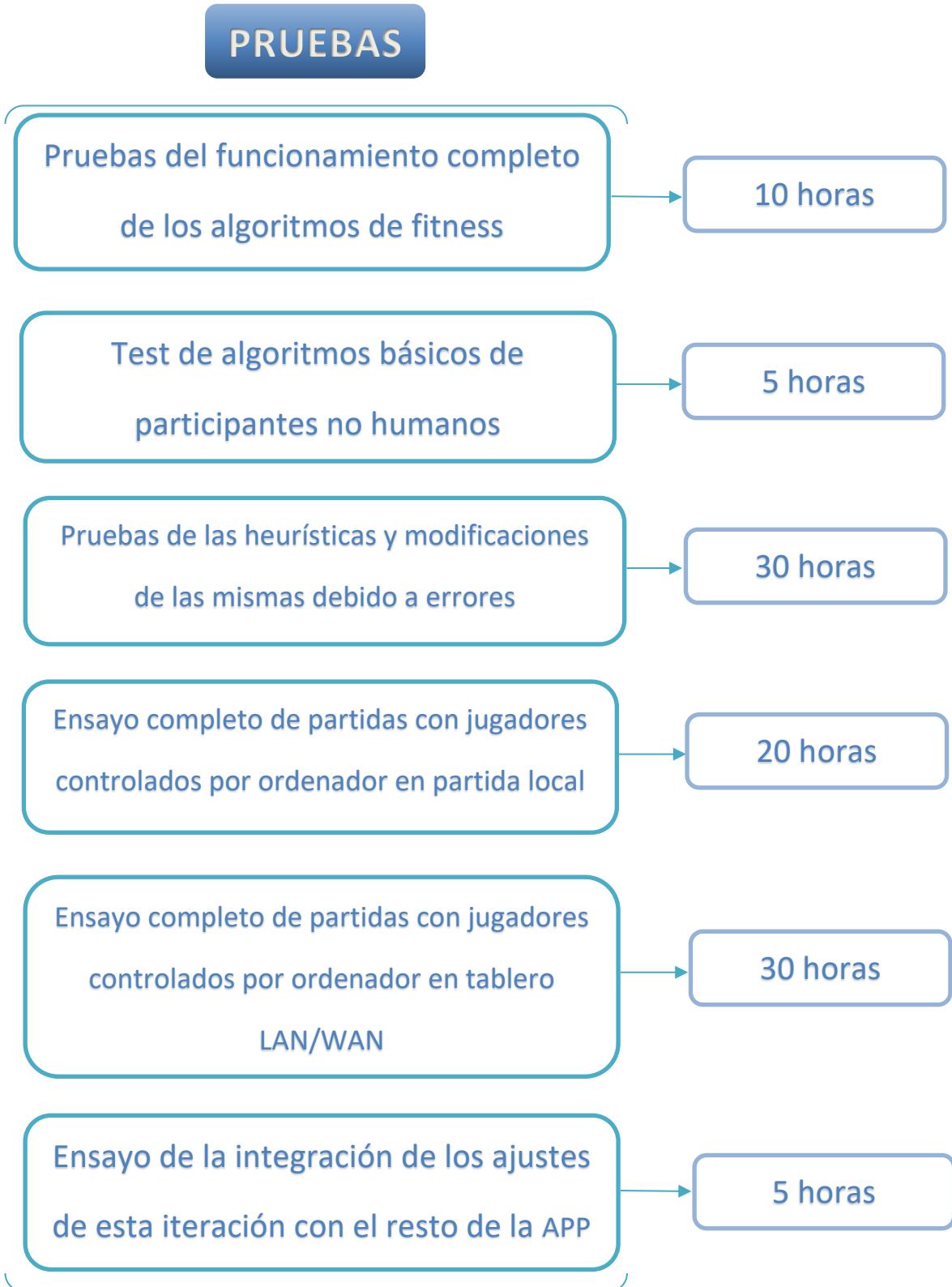
15 horas

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.



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 |

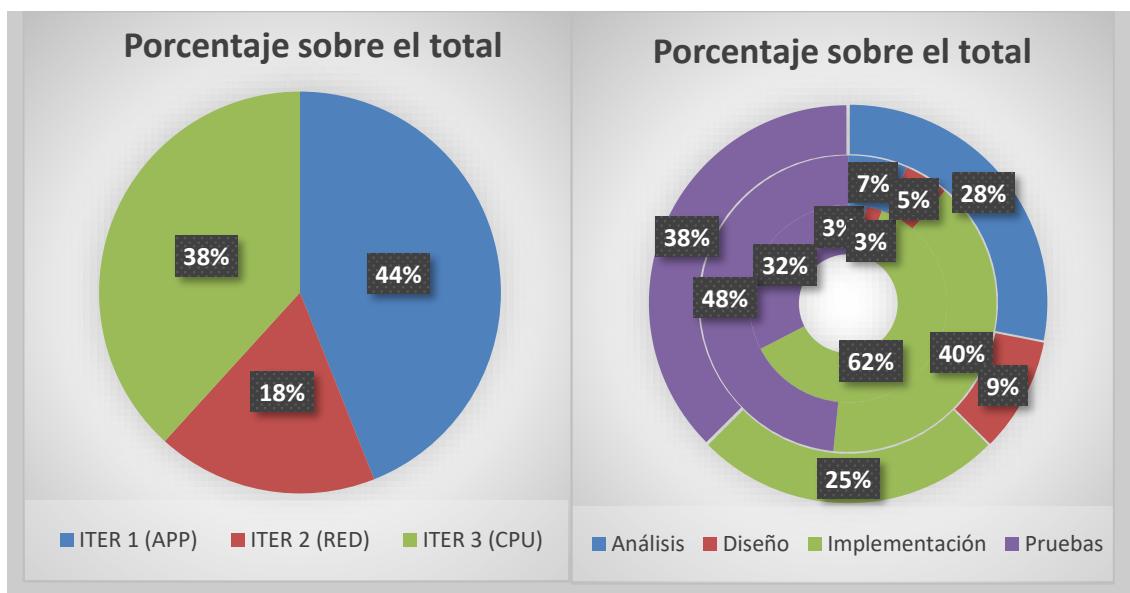


Tabla 2 – División de horas de trabajo por iteración y por etapa.

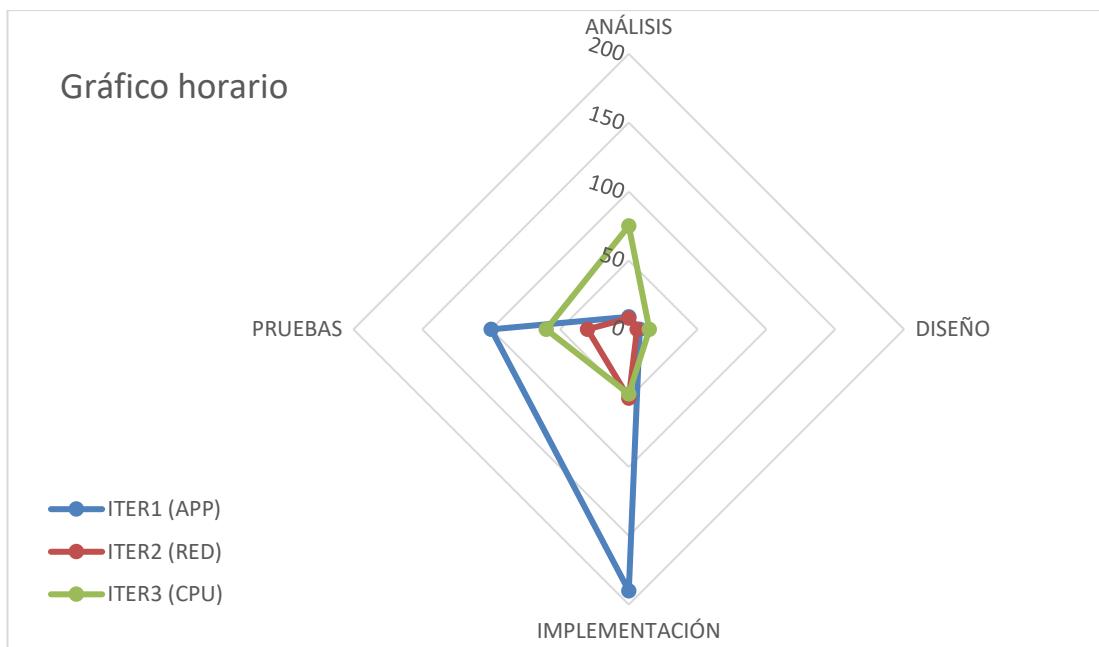


Tabla 3 – Número de horas individuales dedicadas a cada etapa de cada iteración

Pruebas

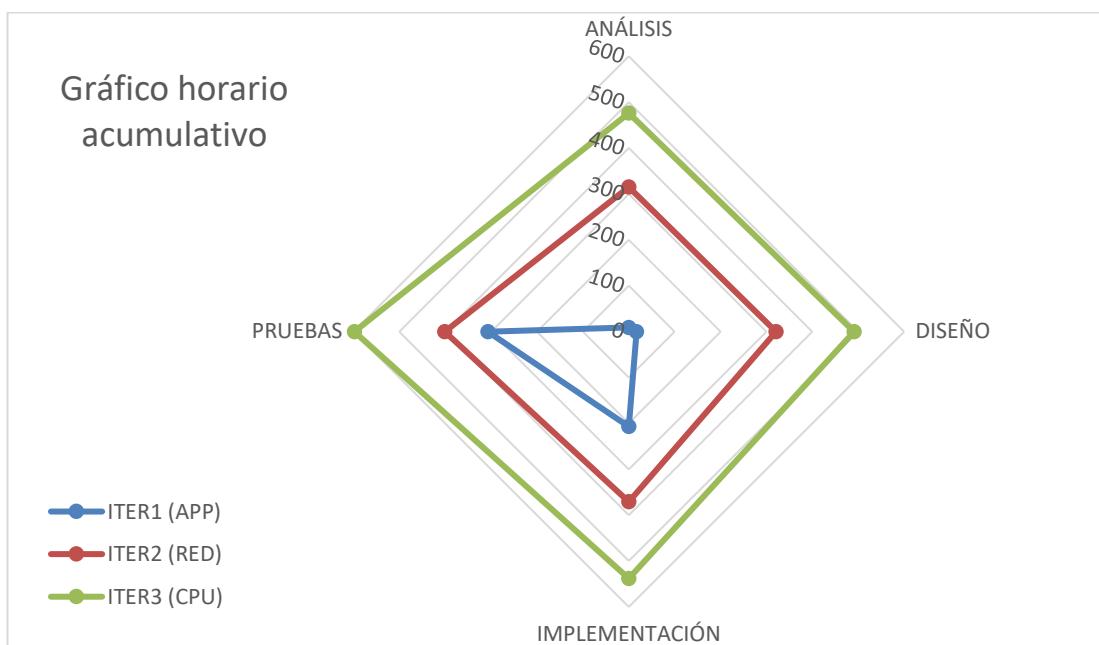


Tabla 4 – Número de horas acumulativas dedicadas

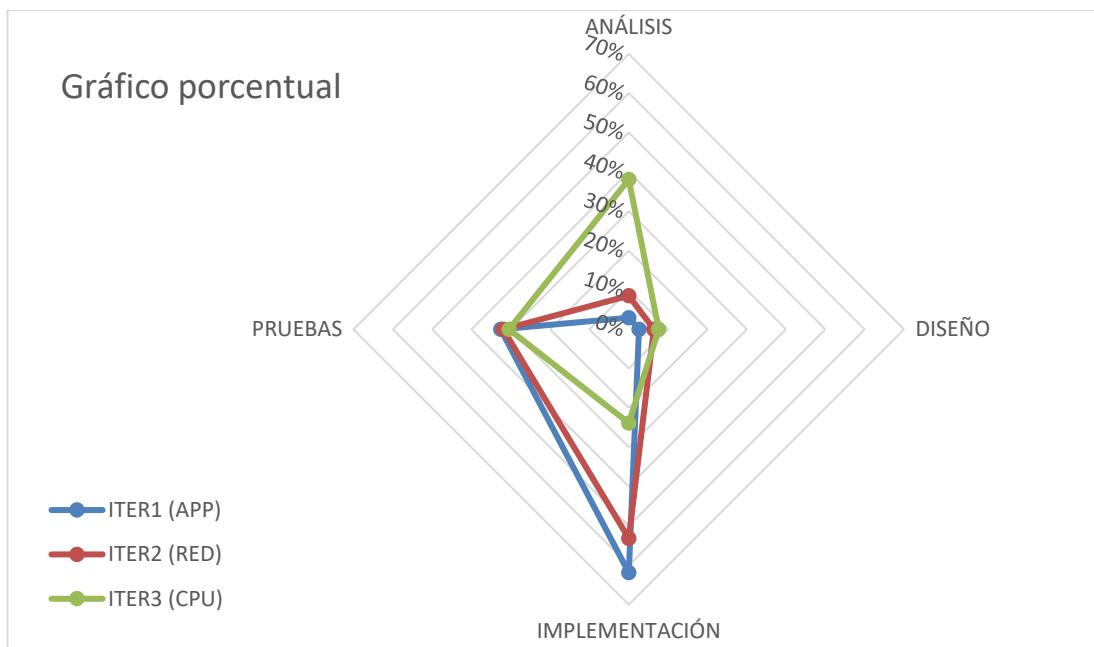


Tabla 5 – Porcentajes de cada etapa sobre el total de cada iteración

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, la librería de elección para nuestra aplicación, es una capa (wrapper) en **Python** para **SDL** (**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.

SPRITE

pygame.sprite.Sprite

- Diseñada para usarse como clase base para los objetos del juego. Para comportamientos específicos requiere otra clase que la supedite, pero tiene métodos útiles.
- **draw()** dibuja el atributo *image* (una instancia de *Surface*) en otra *Surface* destino.
- Varias funciones de **colisión** que detectan cuando los objetos del juego se cruzan entre ellos, usando el atributo *rect* (una instancia de *Rect*)

SURFACE

pygame.Surface

- Controla la representación gráfica de una imagen. Su atributo principal es una imagen cualquiera descomprimida, con métodos que devuelven información como el tamaño y el formato de color de la misma (*transparencia, profundidad de bit...*).
- Mediante la clase auxiliar **pygame.transform**, podemos modificar atributos de una instancia de *Surface*, como su tamaño o rotación.

RECT

pygame.Rect

- Se usan para almacenar áreas rectangulares, que normalmente representan el objeto en la aplicación. Con información como un tamaño y posición, su mayor utilidad es la detección de colisiones y el dibujado de una superficie en otra con bordes limitantes.
- Con métodos para mover y alinear una instancia de esta clase.

Tabla 6 – Clases de **pygame** usadas en el ámbito gráfico

EVENT

pygame.event.Event

- Utilizado para crear nuestros eventos personalizados, y detectar otros ya implementados (como las pulsaciones de teclado, el movimiento del ratón...)
- Para enviar y recoger objetos de este tipo, se dispone de una cola de eventos.
- Cada evento dispone de un entero para indicar el tipo de entero, y un diccionario con el resto de atributos.

MIXER

pygame.mixer

- Contiene clases para cargar objetos sonido en memoria y controlar su reproducción.
- Este módulo inicia cada sonido en un canal distinto, estando estos limitados.
- De serie tenemos 8 canales de sonido, pero el número de estos y su control pueden ser modificados.
- Al reproducir más de un sonido en canales distintos, estos se mezclan en la escucha.

MUSIC

pygame.mixer.music

- Parecido al módulo Mixer. Es utilizado para controlar la reproducción de música en el mezclador de sonido.
- La diferencia principal entre ambos módulos es que *music* no carga el archivo a reproducir de una vez, si no en tiempo real.
- Solo soporta una ejecución simultánea (un objeto sonando).

TIME

pygame.time

- Contiene métodos para programar la ejecución de funciones con una periodicidad temporal X, o para pausar el programa.
- La clase Clock (pygame.time.Clock) sirve para controlar la frecuencia de imagen (framerate) de la aplicación en cuestión. Así conseguimos una ejecución más estable y menos pesada.

Tabla 7 – Clases de pygame que controlan las interacciones del usuario y el resto del entorno (no gráfico)

pygame.display

- Módulo que controla la pantalla principal de pygame. Dicha pantalla, una vez inicializada, es un objeto de tipo *Surface*.
- El objeto resultante puede ser sobreescrito como otra instancia cualquiera, pero para mostrar los cambios se debe actualizar la pantalla real.

pygame.font

- Renderiza fuentes en una instancia de *Surface*

Tabla 8 – Clases menores de pygame utilizadas

LIBRERÍAS EXTERNAS

GRADIENTS

- Dibuja gradientes de un color a otro en distintas formas geométricas y direcciones.

BEZIER

- Calcula los distintos puntos de una curva de bezier para poder dibujarla en pantalla.

PTEXT

- Proporciona gran cantidad de efectos a la renderización de fuentes de pygame (sombras, rotaciones...).

PADLIB

- Capa sobre pygame que proporciona acceso a distintos efectos y rutinas gráficas.

MASTERMIND

- Se ocupa de toda la lógica de nivel bajo en una comunicación entre agentes mediante sockets.
- Permite tener que implementar solo la capa de aplicación adaptada a tu programa.

Tabla 9 – Librerías externas auxiliares

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

Tabla 10 – 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.

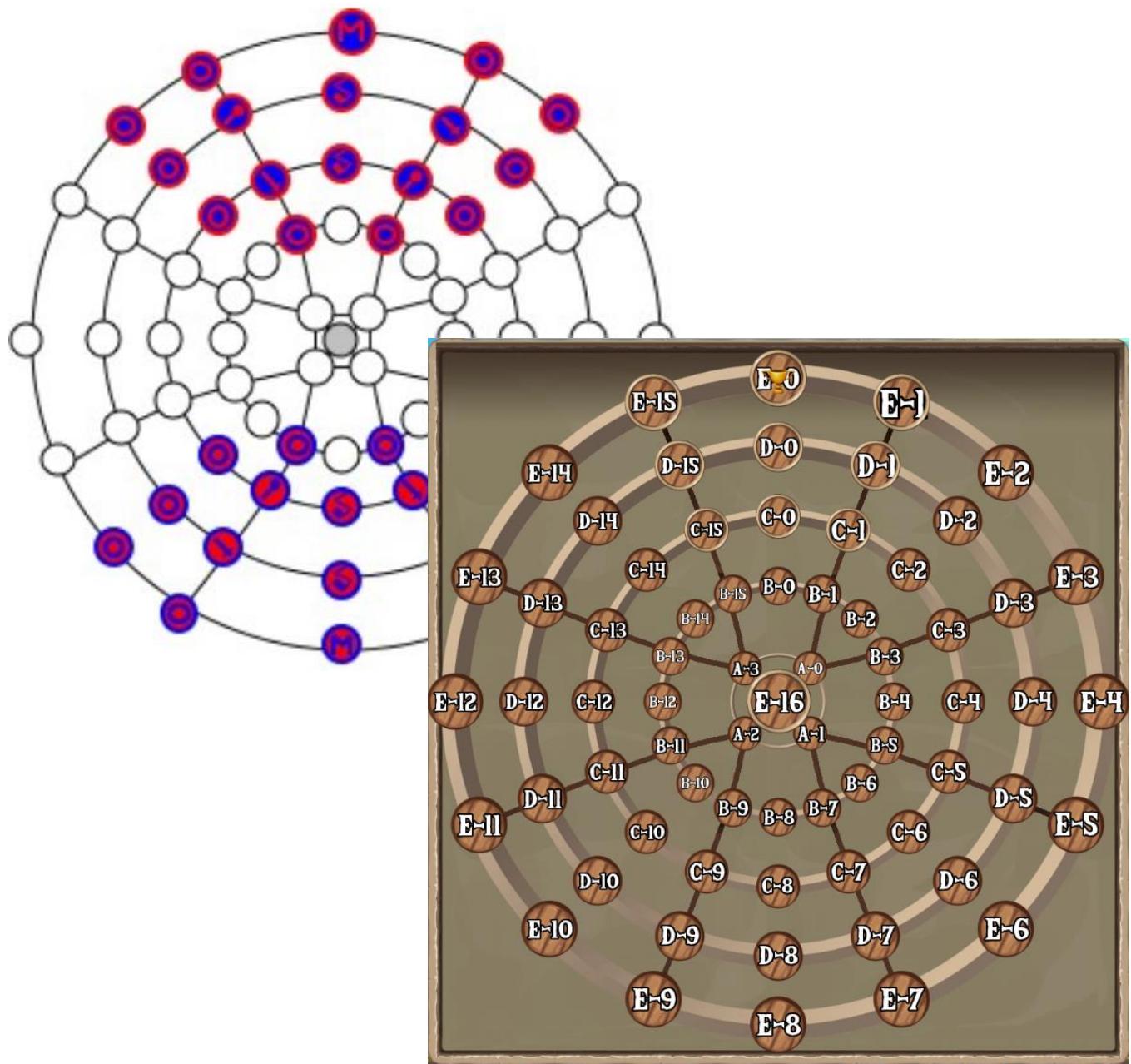


Ilustración 22 – Comparación entre esquema previo y resultado final

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.**



Orco

- Movimiento: **1 espacio**, y debe ser acercándose a un enemigo si es posible (Si existen enemigos en los caminos directos).
- Valor: **1**



Guerrero

- Movimiento: **1 espacio**. No puede volver a la posición anterior. Puede moverse durante **2 turnos** (2 veces).
- Valor: **3**



Sacerdotisa

- Movimiento: **Espacios ilimitados**, a lo largo de un camino conectado, siempre que **no haya piezas intermedias**.
- Valor: **5**



Campeón

- Movimiento: Rango de movimientos de un **mago** y una **sacerdotisa** combinados. Inicialmente **no puede capturar ni ser capturado**.
- Valor: **4 → 15**



Mago

- Movimiento: **3 espacios**. Dan igual piezas intermedias y cambios de dirección (Siempre que no vuelvan)
- Valor: **8**



Matrona

- Movimiento: **1 espacio**. Sin restricciones. **Pieza esencial**, si es capturada, este jugador pierde.
- Valor: **50**

Tabla 11 – 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.



Casilla. Se pueden mover personajes a ella, y reacciona al colisionar con el ratón. Cada una tiene su propio índice.



Casilla de promoción. Contiene un efecto especial si un **Orco** o un **Campeón** aterriza aquí.

Orco, si hay algún personaje de mayor valor capturado, se puede intercambiar por el Orco. Campeón, éste adquiere la capacidad de capturar y ser capturado (aumentando su valor).



Dado. Se permite su uso una vez en cada turno, y un número máximo de veces por partida.

Si tiene como resultado el valor máximo del dado, en ese turno puedes controlar un personaje enemigo. En caso contrario se pierde el turno.



Botón de ayuda. Cuando está activado, el botón central del ratón muestra diálogos con información sobre el elemento que esté seleccionando el cursor.



Botón de información. Se muestra en pantalla la puntuación calculada de cada movimiento posible para el personaje seleccionado (Si está activado).

Tabla 12 – 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=1BgkLwn66qN0oej3QEnijcF-4LunZpjKTA62uQIETTBw>

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 Elistraee 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 requisitos 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 desellejar 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.

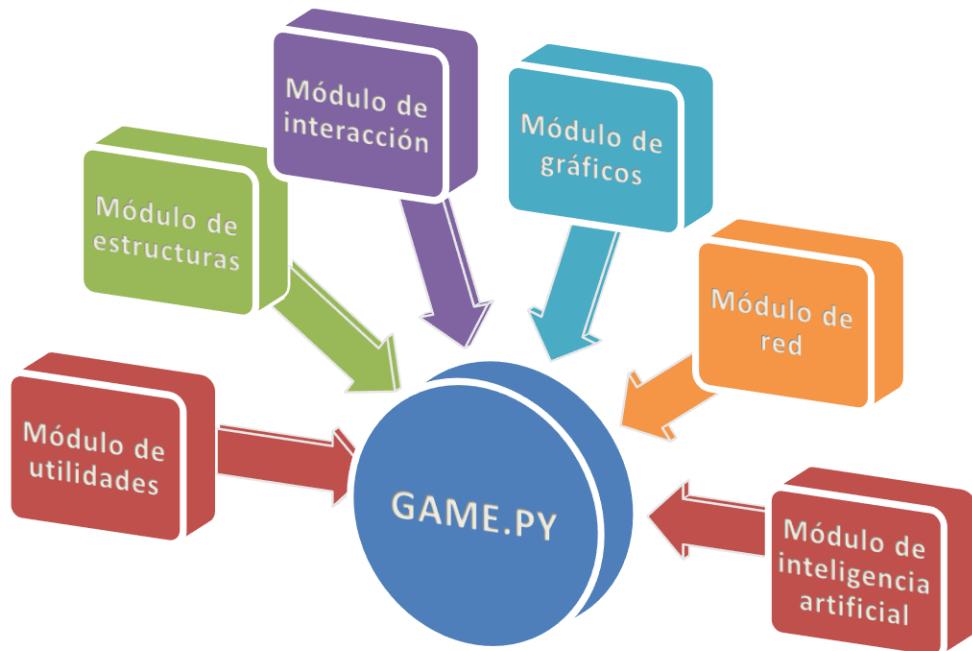


Ilustración 23 – Esquema simplificado de la estructura de la aplicación

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

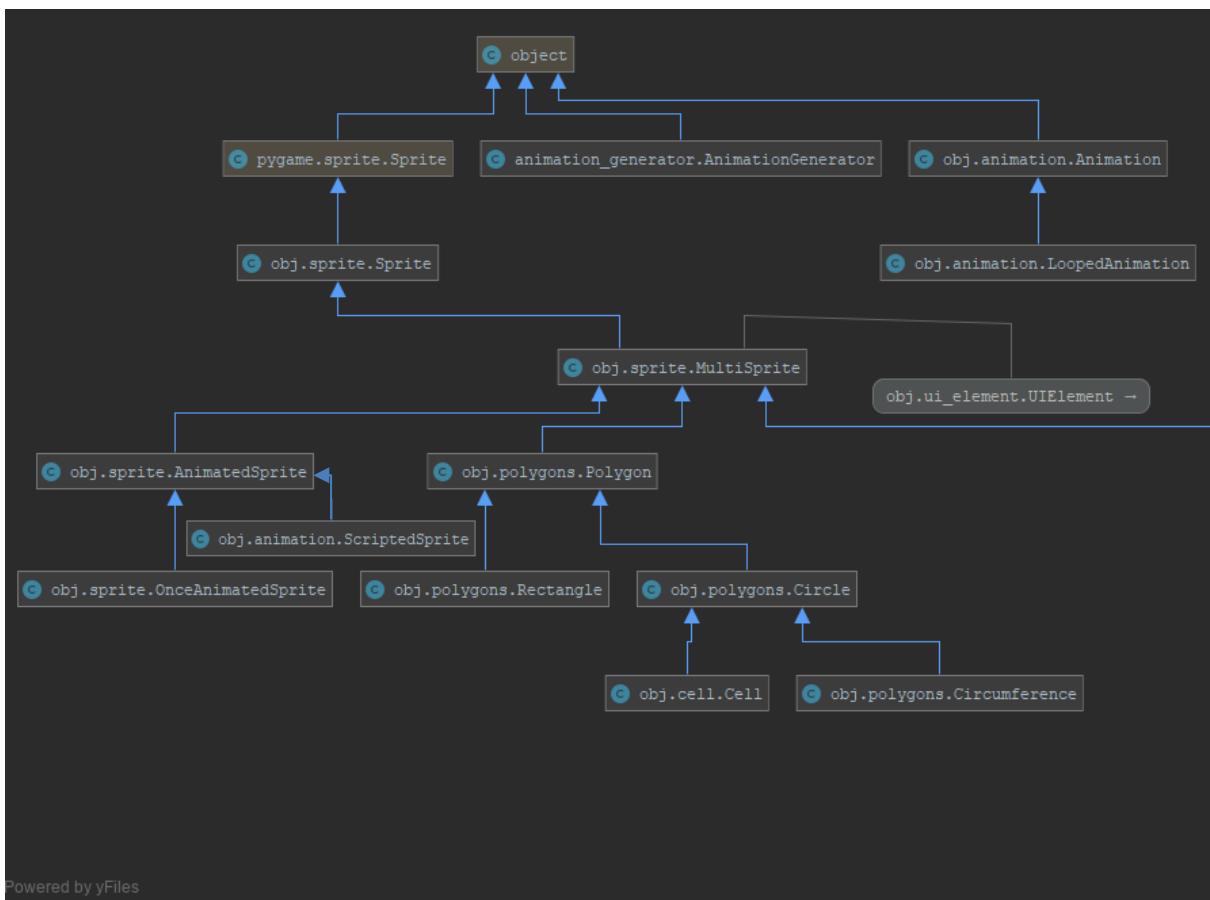


Ilustración 24 – Diagrama UML del módulo gráfico (Excepto UIElement)

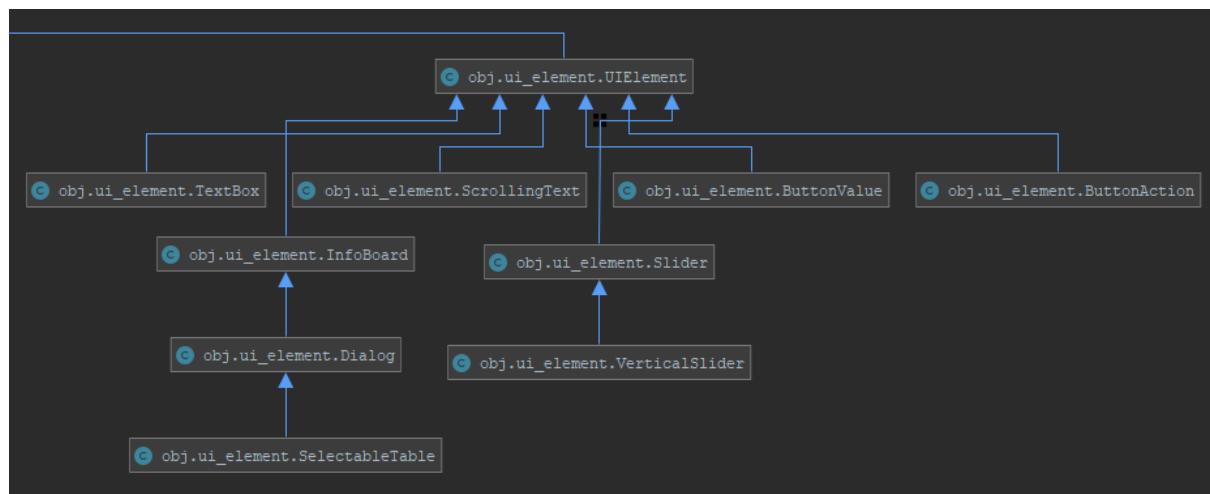


Ilustración 25 - Diagrama UML del módulo gráfico (Solo UIElement)

Sprite**obj.sprite.Sprite**

- Hereda de la clase **Sprite** que te incluye Pygame. Añade atributos y métodos que permiten la generación de la imagen en base a unos parámetros de entrada, el redimensionado de la misma sin perder información, el dibujado de otra textura que actúa de overlay, animaciones cuando esté activo, y más.

MultiSprite**obj.sprite.MultiSprite**

- Hereda de nuestra clase **Sprite**.
- Contiene múltiples sprites y superficies que forman objetos interactivos complejos, mostrándose en pantalla en el orden deseado.

AnimatedSprite**obj.sprite.AnimatedSprite**

- Hereda de **MultiSprite**. Acepta imágenes que usará como frames o fotogramas, avanzando en ellos en cada dibujado en pantalla. Se crea así una animación.

ScriptedSprite**obj.animation.ScriptedSprite**

- Subclase de **AnimatedSprite**. Puede tener programar un movimiento que conseguirá que, en un tiempo estimado, el objeto se mueva de un punto A a otro punto B.
- En resumen, son componentes de animaciones preprogramadas.

UIElement**obj.ui_element.UIElement**

- Subclase de **MultiSprite**. Clase base para implementar interacción con el usuario.
- Soporta colisiones y tiene Eventos asociados, que se disparan cuando la instancia recibe alguna acción del usuario.
- Muchas subclases de ésta son los elementos de los menús y tablero, como un botón, un slider, un diálogo, una tabla con información...

Polygon**obj.polygons.Polygon**

- Hereda de **MultiSprite**. Clase base para implementar distintos tipos de polígonos.
- Sus subclases añaden atributos según sus necesidades.
- Implementa ayuda para comprobar colisiones.

Animation**obj.animation.Animation**

- Animación preprogramada. El término en el mundillo sería "Real time cutscene".
- Durá el tiempo que hayamos puesto de entrada, y contiene/controla varios **ScriptedSprite**.

AnimationGenerator**animation_generator.AnimationGenerator**

- Generador de animaciones. Proporciona métodos para realizar dichas animaciones mediante tan sólo parámetros de entrada, y también tiene algunas ya fabricadas, para crear y usar.

Módulo de red

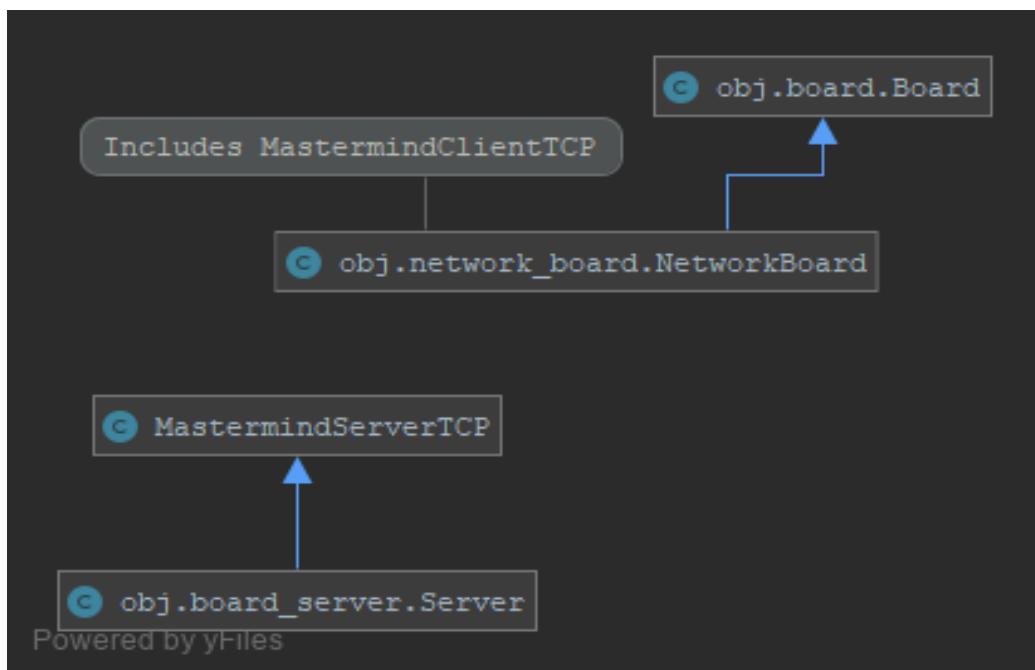


Ilustración 26 – Diagrama UML del módulo de red (Python)

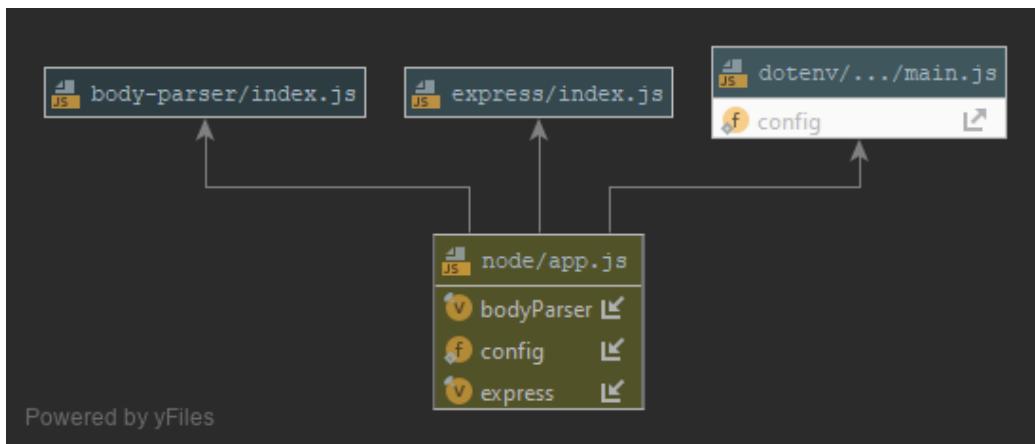


Ilustración 27 – Diagrama UML del módulo de red (JavaScript)

| | |
|---|---|
| Server | obj.board_server.Server |
| <ul style="list-style-type: none"> • Servidor de una partida entre usuarios. • Maneja los mensajes y los datos entre los usuarios, controlando a su vez sus conexiones. | |
| NetworkBoard | obj.network_board.NetworkBoards |
| | <ul style="list-style-type: none"> • Subclase de Board. Es un tablero con capacidades online que actúa como Cliente en nuestro esquema de red. |
| App (Node) | node/app.js |
| <ul style="list-style-type: none"> • Servidor de Express que actúa como la tabla de servidores. • Permite la existencia de servidores públicos. Servicio externo. | |

Tabla 14 – Clases clave del módulo de red

Módulo de inteligencia artificial

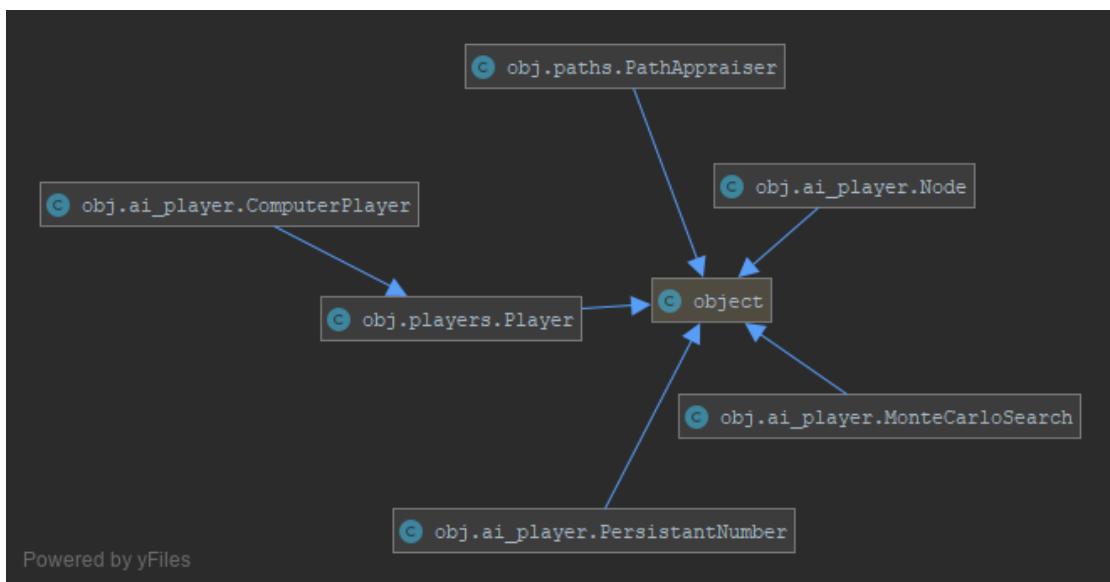


Ilustración 28 - Diagrama UML del módulo de IA

ComputerPlayer

obj.ai_player.ComputerPlayer

- Subclase de **Player**. Tiene todas las acciones que puede realizar un jugador (de la superclase), y añade cómputo y decisiones sobre el tablero y el mapa en el que se esté desarrollando la ejecución.
- Contiene los algoritmos heurísticos y la lógica de decisión de movimientos.

PathAppraiser

obj.paths.PathAppraiser

- Clase auxiliar con métodos que modifican los caminos y tableros de entrada. Estas modificaciones son necesarias para que las heurísticas funcionen correctamente.
- Otra de sus funciones es la puntuación de movimientos en el tablero. Resumidamente, contiene el algoritmo de **fitness**.

Node

obj.ai_player.Node

- Un nodo contenedor de un estado del tablero, incluyendo los jugadores, la posición de los personajes, los ajustes de creación del mapa, y los hashes correspondientes.
- Cada nodo de un árbol en una búsqueda de MonteCarlo es de este tipo.

MonteCarloSearch

obj.ai_player.MonteCarloSearch

- Posee todos los métodos y funciones para realizar la heurística de montecarlo, con la heurística en sí también.

Tabla 15 - Clases clave del módulo de IA

Módulo de interacción (lógica)

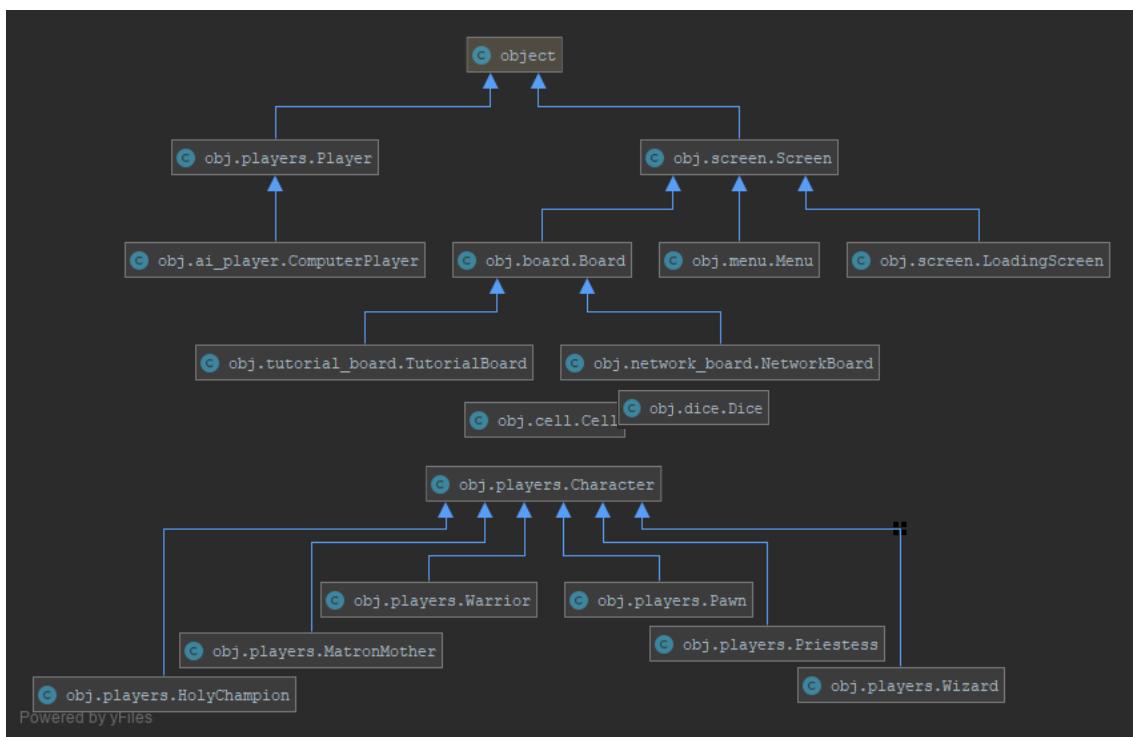


Ilustración 29 - Diagrama UML del módulo de interacción

Cell

`obj.cell.Cell`

- Subclase de **Circle**.
- Cada casilla del tablero pertenece a este tipo de objeto.
- Puede hospedar personajes en sus instancias.
- Algunas tienen la propiedad de "promocionar" personajes.

Dice

`obj.dice.Dice`

- Hereda de **AnimatedSprite**.
- Es el dado utilizado en la pantalla inicial para elegir los turnos, y durante la partida para intentar controlar personajes enemigos.
- Se pueden cambiar el intervalo de valores mediante los parámetros de entrada.

Character

`obj.players.Character`

- Otra subclase de **AnimatedSprite**.
- Añade toda la funcionalidad propia de un personaje del tablero. Caminos posibles, restricciones de movimiento, etc.

Subtype of Character

`obj.players.*`

- Los 6 tipos de personaje todos heredan de **Character**. Esencialmente, cambian entre ellos en las restricciones de movimiento y el atributo valor.

Tabla 16 - 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.

Screen

`obj.screen.Screen`

- Su propósito es controlar las interacciones y elementos comunes a todas las pantallas de la aplicación.
- A su vez es una superclase para varias subclases importantes.
- Los eventos del usuario son manejados aquí antes de pasar a las subclases que los necesiten.
- Entre esos elementos comunes a controlar se incluye el mostrado en pantalla y la inclusión de animaciones.

LoadingScreen

`obj.screen.LoadingScreen`

- Subclase de **Screen**.
- Sus atributos están diseñados para formar una pantalla de carga para mostrar.

Menu

`obj.menu.Menu`

- Hereda de **Screen**.
- Añade instancias de **UIElement** o alguna de sus subclases, con las que el usuario puede interaccionar para conseguir una respuesta.

Board

`obj.board.Board`

- Su superclase es **Screen**. Es el tablero que engloba todos los componentes y lógica que hace posible el funcionamiento de una partida.
- Controla los jugadores, los personajes, los eventos del usuario...
- Imprescindible, ya que sin esta clase, no hay aplicación.

TutorialBoard

`obj.tutorial_board.TutorialBoard`

- Subclase de **Board**. Realizada con el único objetivo de tener un tipo de partida para principiantes, que todavía no conocen las normas de un duelo al juego.
- Sus diferencias son la aparición de mensajes de ayuda, la cantidad de personajes y su variedad.

Módulo de estructuras

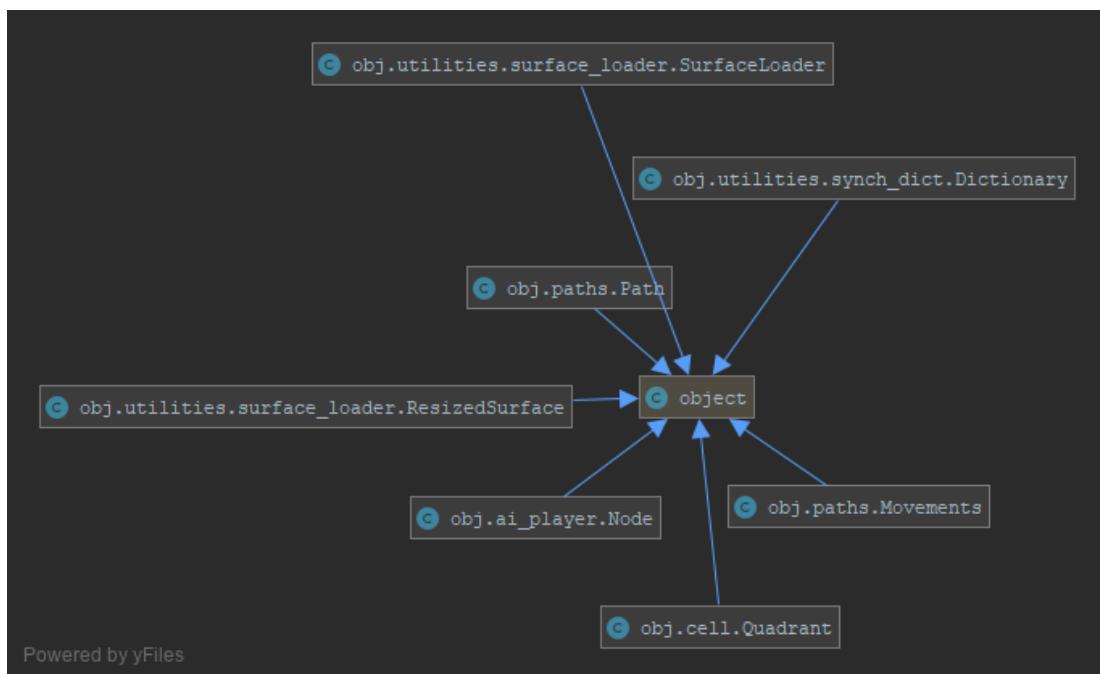


Ilustración 30 - Diagrama UML del módulo de estructuras

Dictionary

`obj.utilities.synch_dict.Dictionary`

- Subclase de **Dict**.
- Diccionario con control sobre el acceso de múltiples hilos simultáneamente.

Quadrant

`obj.cell.Quadrant`

- Contenedor inicial de casillas del tablero (**Cell**).
- Sólo se utiliza en un comienzo, para clasificar las casillas en cuadrantes.

Path

`obj.paths.Path`

- Objeto resultante cuando se realiza la conversión de una casilla a otro tipo de objeto más ligero. Así las estructuras a manejar no cambian por otros procesos, y ocupan menos espacio.

Movements

`obj.paths.Movements`

- Contenedor de todos los caminos posibles, desde cualquier casilla, para todos los personajes, en un tablero concreto. Se va ocupando por demanda.
- Contiene métodos para devolver e insertar estos caminos.

SurfaceLoader

`obj.utilities.surface_loader.SurfaceLoader`

- Diccionario de hashes, contenedor de las imágenes sin modificar cargadas hasta ahora.
- Contiene métodos para devolver e insertar estas instancias de **Surface**.

ResizedSurface

`obj.utilities.surface_loader.ResizedSurface`

- Diccionario de hashes, contenedor de las imágenes redimensionadas/modificadas cargadas hasta ahora.
- Contiene métodos para devolver e insertar estas instancias de **Surface**.

Tabla 18 - Clases clave del módulo de estructuras

Módulo de utilidades

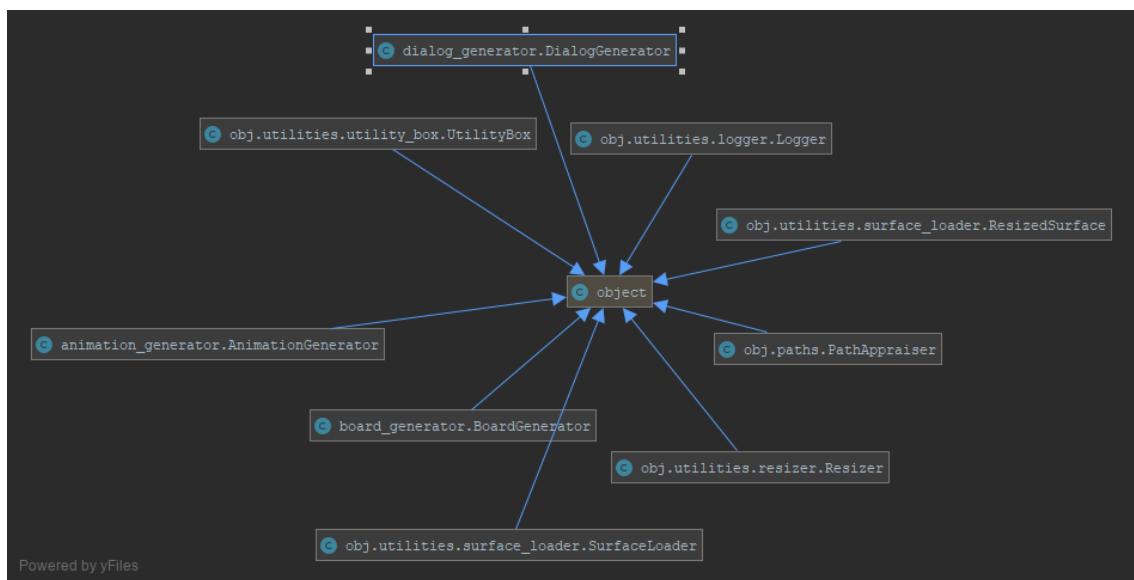


Ilustración 31 - Diagrama UML del módulo de utilidades

Resizer

`obj.utilites.Resizer`

- Reúne toda la lógica que usa la aplicación para redimensionar y modificar imágenes/**Surfaces**.

UtilityBox

`obj.utilites.UtilityBox`

- Clase miscelánea que **junta todos los métodos/funciones auxiliares** que o bien, no tienen cabida en otra clase, o bien no tienen el peso/complejidad suficiente para tener su propia clase.

Logger

`obj.utilites.Logger`

- Muestra sucesos por la línea de comandos con distintos mensajes, según la prioridad de los mismos.
- Guarda en un archivo local toda la información de una ejecución.

Generators

`/* */`

- BoardGenerator crea un tablero según los ajustes que el usuario tenga configurados en el menú correspondiente.
- El resto son generadores de distintas instancias prefabricadas de distintos objetos.

Tabla 19 - 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 DEL CÓDIGO RESULTANTE

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

TODO

ESTADÍSTICAS DE LOS ARCHIVOS LOCALES RESULTANTES

Líneas código con y sin comentarios y porcentajes sobre el total...

TODO Cuando arreglar bugs

| Módulo | Archivo | Líneas de código | Líneas sin comentarios ni espacios | Porcentaje Módulo/Total |
|--------|---------|------------------|------------------------------------|-------------------------|
| | | | | |

Tabla 20 – Tabla de ficheros .py, ordenados por módulo

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

| Módulo | Archivo | Líneas de código | Líneas sin comentarios ni espacios | Porcentaje Módulo/Total |
|--------|---------|------------------|------------------------------------|-------------------------|
| | | | | |

Tabla 21 – Tabla de ficheros .py, ordenados por líneas de código

| TIPO DE FICHERO | NÚMERO DE FICHEROS | TAMAÑO EN DISCO | TAMAÑO ACUMULATIVO |
|-----------------|--------------------|-----------------|--------------------|
| IMÁGENES | | | |
| AUDIO | | | |
| CÓDIGO PYTHON | | | |
| OTROS | | | |

Tabla 22 – Tabla de tamaño conjunto de ficheros por tipo, ordenados por tamaño total

Estructuras de datos

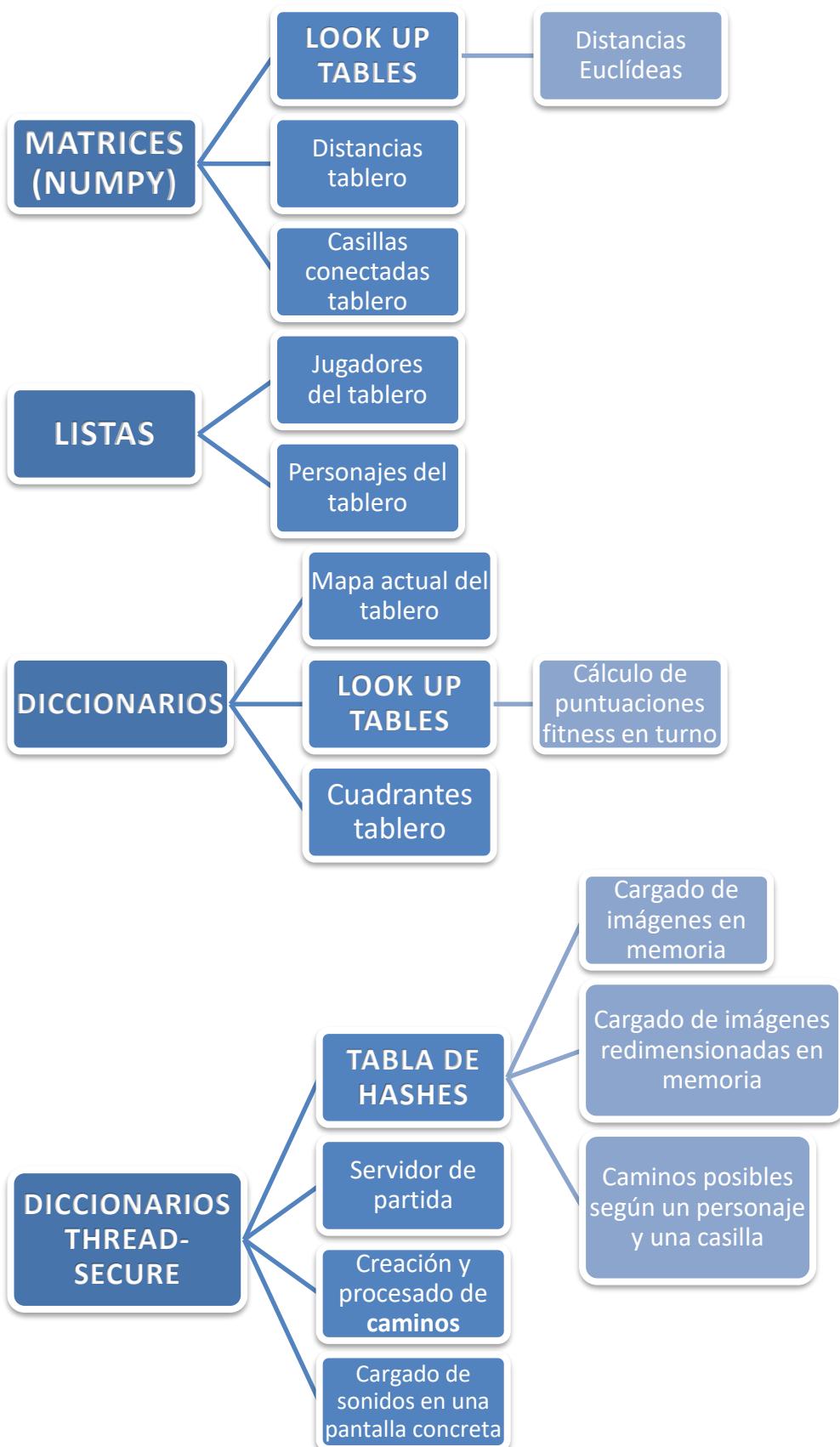


Tabla 23 – Estructuras de datos utilizadas y sus propósitos

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.

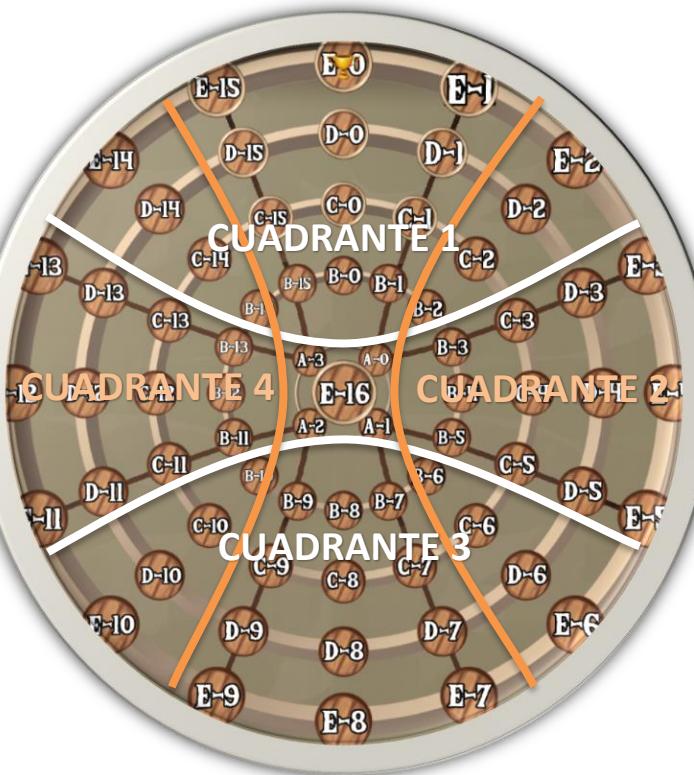


Ilustración 24 – Ejemplo de cuadrantes en un tablero

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 25 – Pseudocódigo de distancias euclídeas

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

Tabla 26 - 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 27 - 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 28 - Pseudocódigo de cálculo de puntuaciones de movimiento en el tablero

| <i>Método</i> | Tiempo de cómputo | Ahorro medio |
|-------------------------------------|-------------------|--------------|
| <i>Generación inicial</i> | | |
| <i>Generación en cada selección</i> | | |

Tabla 29 - 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.

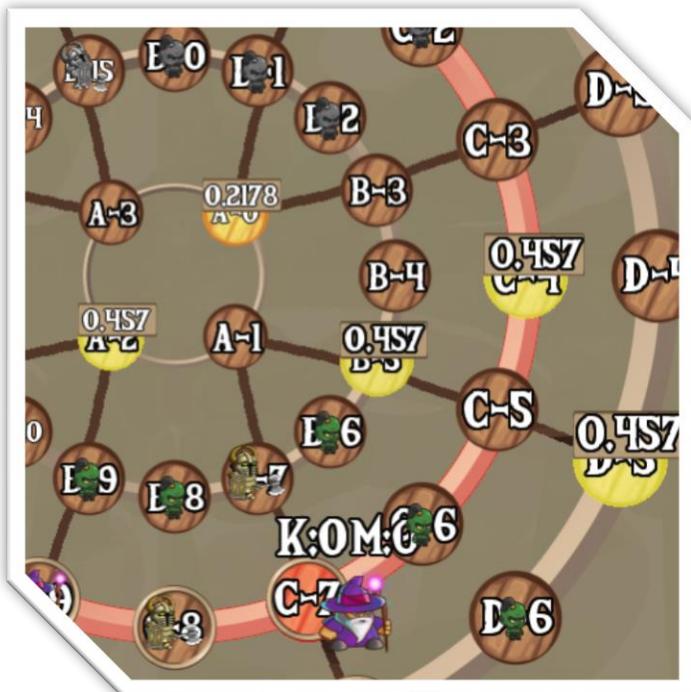


Ilustración 32 – Ejemplo de puntuaciones ‘fitness’ en el tablero

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 30 - Lut inicial para los caminos de un tablero

| Método | Tiempo de cómputo | Ahorro medio |
|------------------------------|-------------------|--------------|
| Generación inicial | | |
| Generación en cada selección | | |

Tabla 31 - 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.

Entero: Espacios por movimiento

Booleano: Puede saltar aliados

Booleano: Puede saltar enemigos

Booleano: Solo caminos continuos

Ilustración 33 – 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 32 – Pseudocódigo de la estructura LUT de imágenes cargadas

| <i>Resolución</i> | Carga Memoria (LUT imágenes) | Carga Memoria (normal) | Ahorro |
|-------------------|---------------------------------|---------------------------|------------------|
| 1920*1080 | 1115 MB | 1790 MB | =(1115/1790)*100 |

Tabla 33 – Ejemplo de uso de memoria según la técnica en uso

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

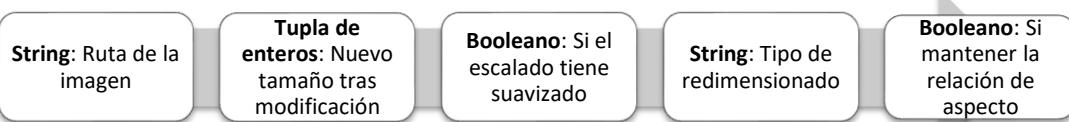
Cada vez que un objeto de tipo Surface o Imagen necesita ser cargado en memoria, ya sea para una instancia de Sprite concreta u otro propósito, se realiza mediante el método arriba mostrado.

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 34 - 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*.



Ilustración 35 - 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 36 – 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.

| <i>Resolución</i> | Gasto | Gasto | Tiempo Carga | Tiempo Carga |
|-------------------|---------|----------|-----------------|-----------------|
| | Memoria | Memoria | Inicial/Tablero | Inicial/Tablero |
| | (LUT) | (Normal) | (LUT) | (Normal) |
| 640*360 | 700 MB | 1315 MB | 5410ms/19460ms | 9700ms/59200ms |
| 848*480 | 715 MB | 1377 MB | 6350ms/18020ms | 9700ms/59000ms |
| 1024*576 | 800 MB | 1437 MB | 7400ms/17420ms | 14400ms/59500ms |
| 1280*720 | 824 MB | 1565 MB | 8570ms/18670ms | 12000ms/63000ms |
| 1366*768 | 910 MB | 1532 MB | 9210ms/21810ms | 13300ms/60000ms |
| 1600*900 | 902 MB | 1632 MB | 10425ms/20120ms | 15400ms/ |
| 1920*1080 | 1115 MB | 1790 MB | 14200ms/23420ms | 20100ms/92-29 |

Tabla 34 – 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).

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) |
|------------------------|---------------------------|--|---|
| 1380 MB | | 14512ms/25600ms | 15286ms/ |

Tabla 35 – Gasto de memoria con el tamaño de las superficies deslimitado (Resolución 1920*1080)

| Resolución | Ahorro memoria | Ahorro tiempo | Ahorro tiempo |
|------------|----------------|---------------|---------------|
| | | Carga Inicial | Carga Tablero |
| 640*360 | | | |
| 848*480 | | | |
| 1024*576 | | | |
| 1280*720 | | | |
| 1366*768 | | | |
| 1600*900 | | | |
| 1920*1080 | | | |

Tabla 36 - Ahorro de memoria y tiempo para todas las resoluciones, usando LUTs

Pruebas de rendimiento

MSI GE63VR-7RF



- Intel I7-7700HQ
- 4 núcleos
- 4 hilos
- 2.8 - 3.8 Ghz
- Nvidia GTX1070
- 16GB DDR4
- 256GB SSD

Thinkpad L460



- Intel I5-6200U
- 2 núcleos
- 4 hilos
- 2.3 - 2.8 Ghz
- Intel HD 520
- 16GB DDR4
- 256GB SSD

Lenovo G50-45



- AMD E1-6010
- 2 núcleos
- 2 hilos
- 1.35 Ghz
- AMD Radeon R2 Graphics
- 8GB DDR3
- 1000GB HDD

Intel PC Stick



- Intel Z3735F
- 4 núcleos
- 4 hilos
- 1.33 - 1.83 Ghz
- Intel HD Graphics
- 2GB DDR3
- 32GB EMMC

| <i>Equipo</i> | <i>Procesador</i> | <i>Consumo</i> | <u>Rendimiento</u> | <i>Precio (Est.)</i> | <i>Relación</i> |
|--------------------|-------------------|----------------|--------------------|----------------------|-----------------|
| <i>MSI</i> | I7 7700HQ | 45W/230W | 8806 pts | 1000€ | 8.8pts/€ |
| <i>Thinkpad</i> | I5 6200U | 15W/45W | 4020 pts | 400€ | 10.0pts/€ |
| <i>Lenovo</i> | E1 6010 | 10W/45W | 862 pts | 180€ | 4.8pts/€ |
| <i>Intel Stick</i> | Atom Z3735F | 4.4W/10W | 908 pts | 60€ | 15.1pts/€ |

Tabla 37 - Comparativa de equipos de prueba

<SAVA DROW>

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

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:

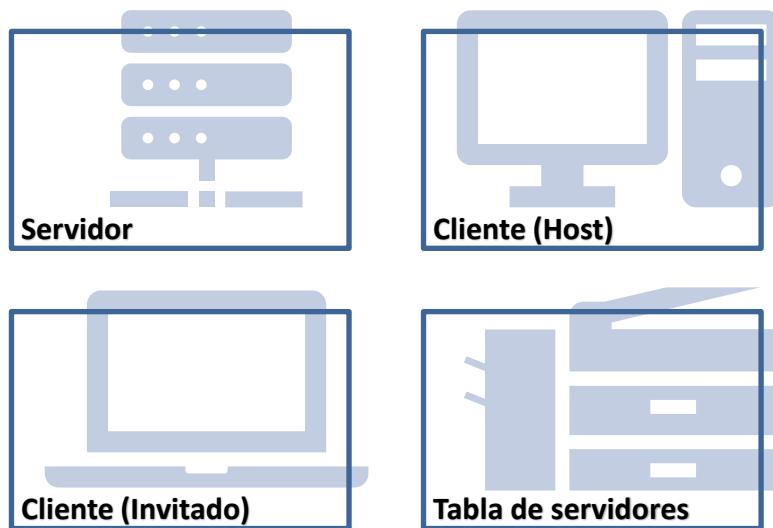


Tabla 38 – Leyenda de los esquemas de red

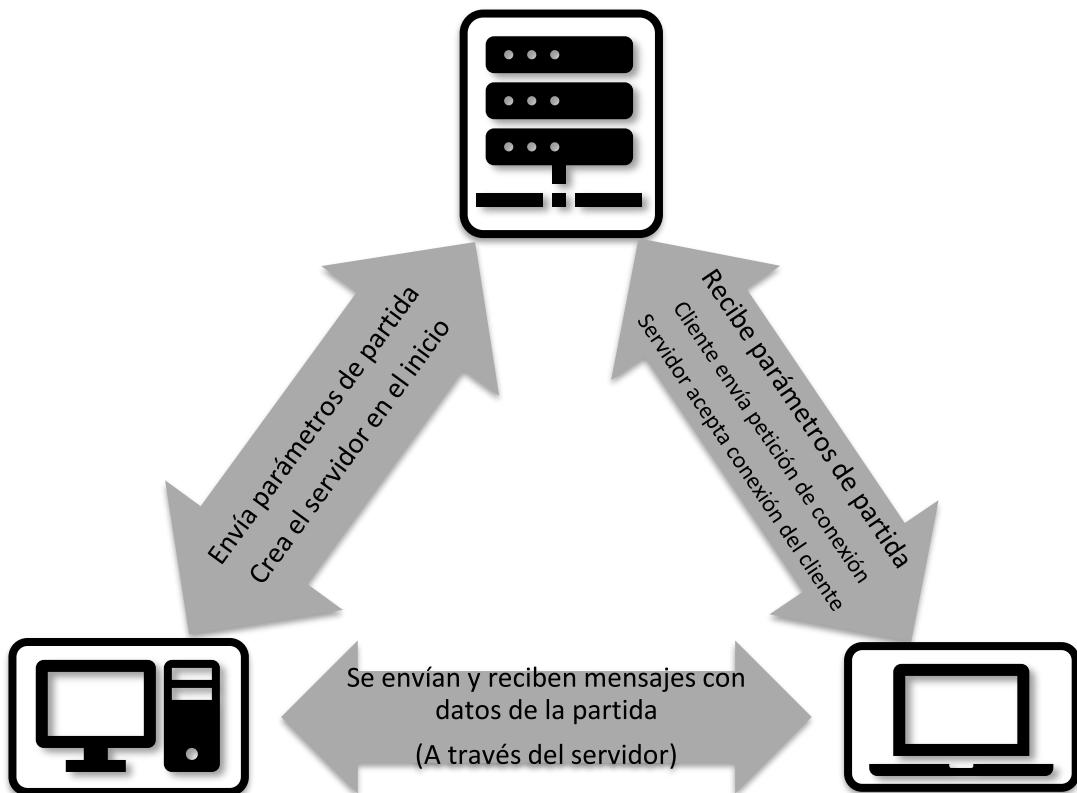


Ilustración 37 – Esquema de comunicación en una conexión a un servidor privado

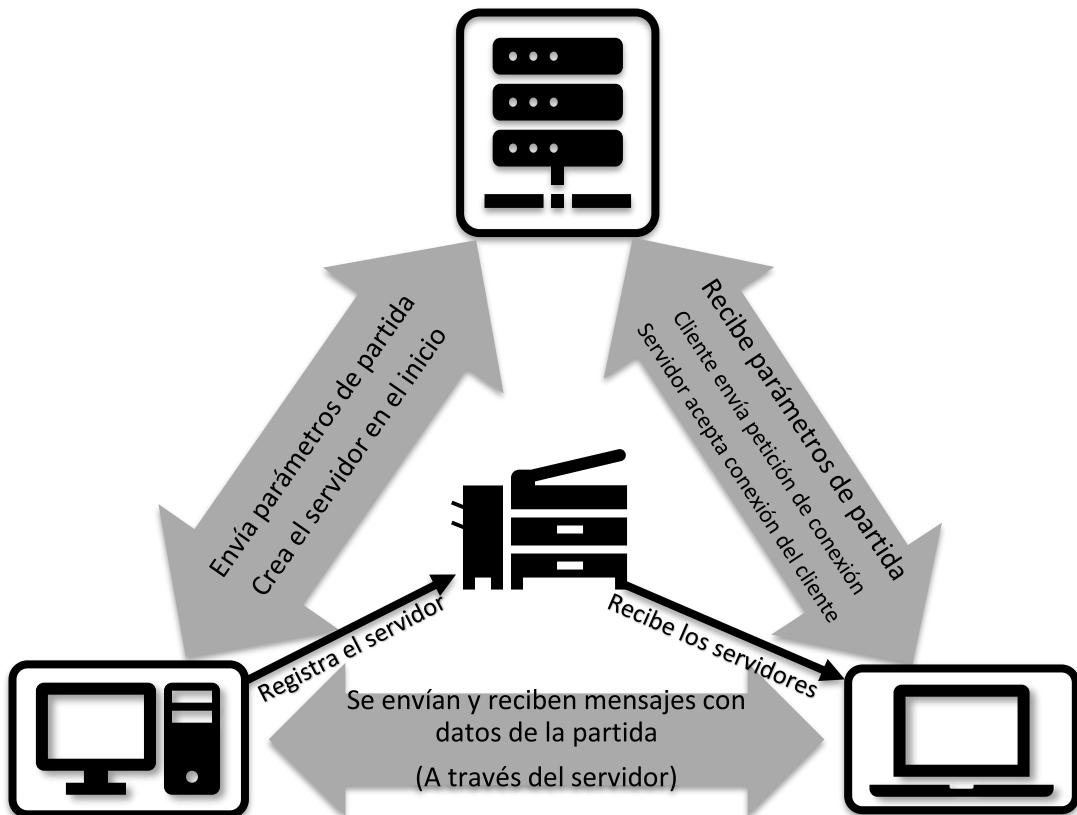


Ilustración 38 – Esquema de comunicación en una conexión a un servidor privado

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**):

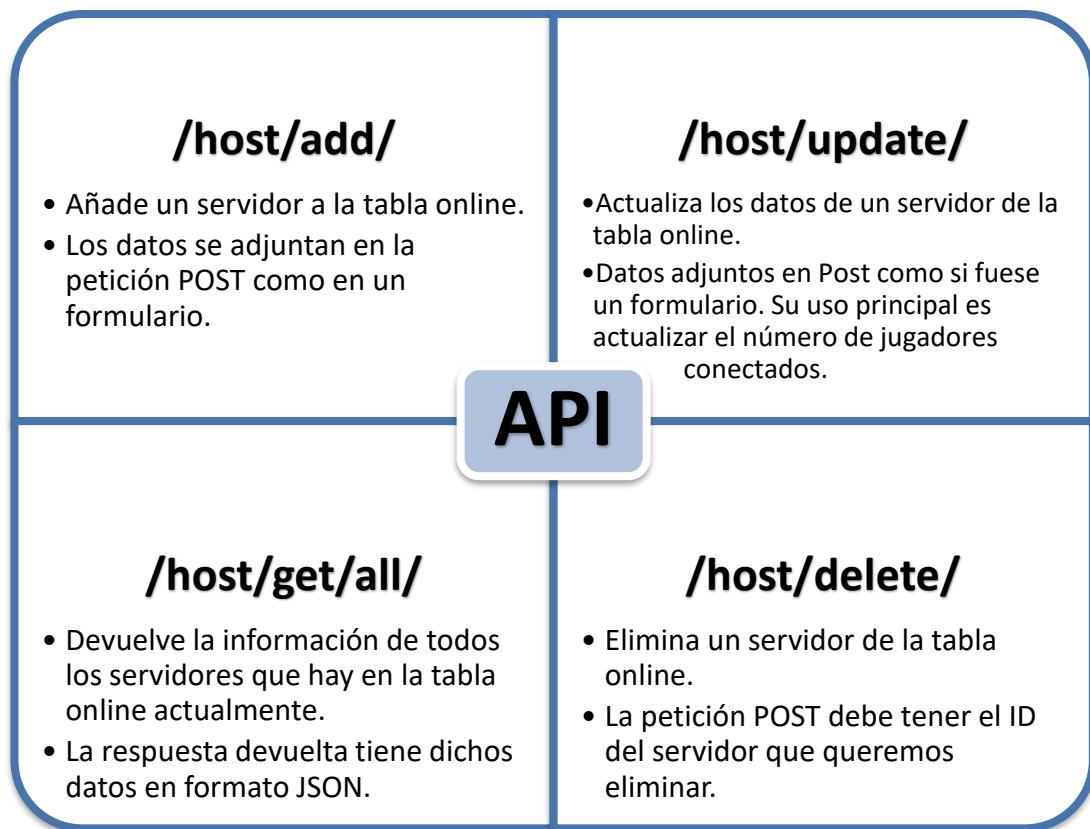


Ilustración 39 – Todos los puntos de acceso del servidor de Express

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 39 – 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.

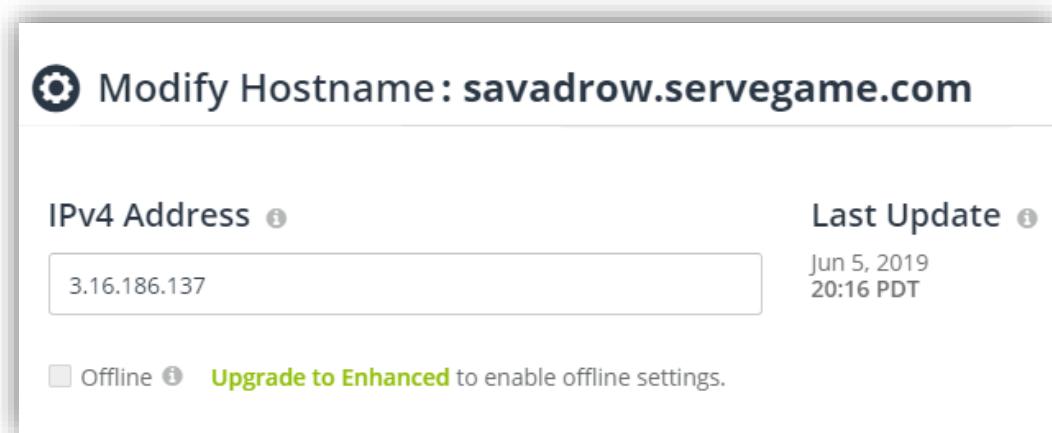


Ilustración 40 – Panel de ajustes de Noip

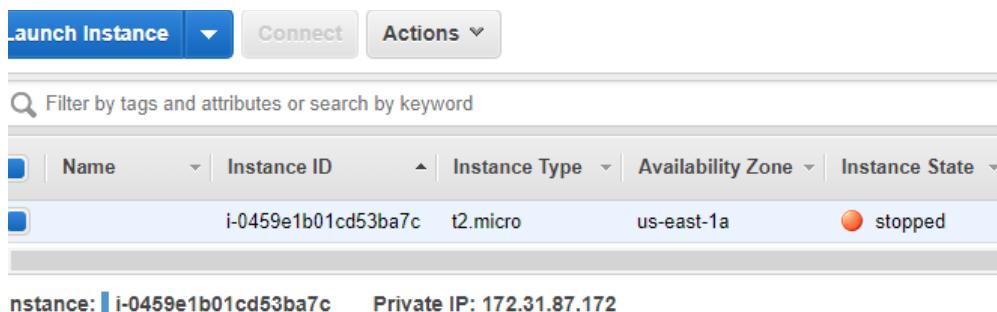


Ilustración 41 – Panel de control de instancias en el servicio web de Amazon (AWS)

TABLA DE MENSAJES

Conexión/Sincronización

| Mensaje | Origen | Destino | Función |
|------------------------|--------------------|--------------------|----------------------|
| <i>host</i> | Cliente (Todos) | Servidor | Registro de conexión |
| <i>start</i> | Servidor | Cliente (Todos) | Comienzo de partida |
| <i>ready</i> | Cliente (Todos) | Servidor | Tablero creado |
| <i>pause</i> | Cliente (Todos) | Cliente (Todos) | Pausar partida |
| <i>params</i> | Cliente (Host) | Cliente (Invitado) | Parámetros partida |
| <i>players</i> | Cliente (Invitado) | Servidor | ¿Cuantos jugadores? |
| <i>start_dice</i> | Cliente (Todos) | Servidor | Valor tirada inicial |
| <i>keep_alive</i> | Cliente (Todos) | Servidor | Mantener la conexión |
| <i>disconnect</i> | Cliente (Todos) | Servidor | Desconectar cliente |
| <i>players_data</i> | Cliente (Host) | Cliente (Invitado) | Datos de jugadores |
| <i>characters_data</i> | Cliente (Host) | Cliente (Invitado) | Datos de personajes |

Tabla 40 – Mensajes usados en la conexión y sincronización de usuarios

Partida en curso

| Mensaje | Origen | Destino | Función |
|-------------------------|-----------------|-----------------|-----------------------|
| <i>swap</i> | Cliente (Todos) | Cliente (Todos) | Promoción personaje |
| <i>admin</i> | Cliente (Todos) | Cliente (Todos) | Modo admin activado |
| <i>end_turn</i> | Cliente (Todos) | Cliente (Todos) | Final de turno |
| <i>turncoat</i> | Cliente (Todos) | Cliente (Todos) | Modo traidor True |
| <i>dice_value</i> | Cliente (Todos) | Cliente (Todos) | Valor de tirada |
| <i>move_character</i> | Cliente (Todos) | Cliente (Todos) | Movimiento personaje |
| <i>drop_character</i> | Cliente (Todos) | Cliente (Todos) | Colocar pj en casilla |
| <i>lock_characters</i> | Cliente (Todos) | Cliente (Todos) | Bloquear personajes |
| <i>mouse_position</i> | Cliente (Todos) | Cliente (Todos) | Movimiento cursor |
| <i>update_character</i> | Cliente (Todos) | Cliente (Todos) | Promoción personaje |

Tabla 41 – 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.



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



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



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



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



- 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.



Tabla 42 – 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.

$$\left(\frac{valor_{mi\ personaje}}{cantidad_{personajes}} \right) * \left(\frac{valor_{mi\ personaje}}{valor_{total\ aliados}} \right) * (99\ si\ matrona\ | 1\ si\ no)$$

Ecuación 2 – Fórmula del multiplicador de peligro

$$\frac{1}{cantidad_{enemigos\ con\ acceso}} * \text{Multiplicador de peligro}$$

Ecuación 3 - 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.

$$1 * \sqrt{\frac{valor_{enemigo}}{valor_{mi\ personaje}}}$$

Ecuación 4 – 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.

$$1 * \left(\frac{mínimo(valor_{enemigos\ con\ acceso})}{valor_{mi\ personaje}} \right) * \sqrt{cantidad_{aliados\ con\ acceso}}$$

Ecuación 5 – Fórmula de la puntuación de cebo

$$0.6 * (\tanh P_{eligo score}) + 0.6 * (\tanh C_{aptura score}) + 0.2 * (\tanh C_{ebo score})$$

Ecuación 6 – 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.

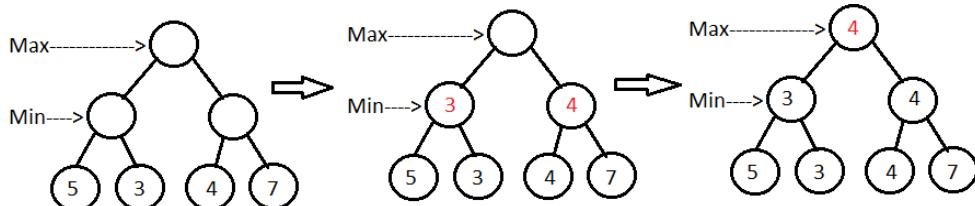


Ilustración 42 – Ejemplo de ejecución Minimax

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.possible_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 43 – Pseudocódigo del algoritmo de poda alfa-beta utilizado

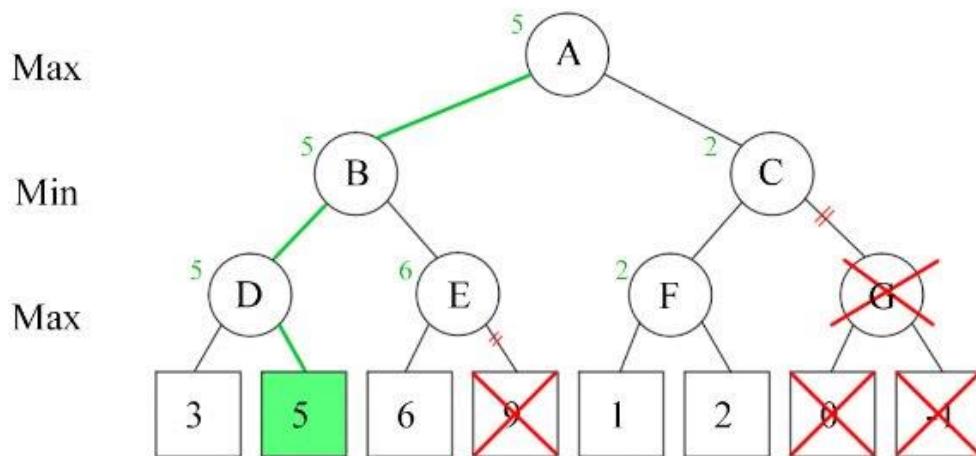


Ilustración 43 – Representación gráfica del árbol resultante tras una ejecución de poda alfa beta

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*

$$UCBT = \left(\frac{\text{Total}_{\text{valor simulaciones}}}{\text{Total}_{\text{visitas nodo}}} \right) + Cte_{\text{exploración}} * \sqrt{\frac{\ln(\text{Total}_{\text{visitas nodo padre}})}{\text{Total}_{\text{visitas nodo}}}}}$$

Ecuación 7 – 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_decision(Hoja)
            valor_tablero_final = Hoja.valor_tablero
        MIENTRAS Hoja NO ES nodo_raíz: #Retropropagación
            Hoja.valor_total += valor_tablero_final
            Hoja.visitas_totales += 1
            Hoja = Hoja.nodo_padre
    
```

Tabla 44 – Pseudocódigo del algoritmo de búsqueda MonteCarlo utilizado

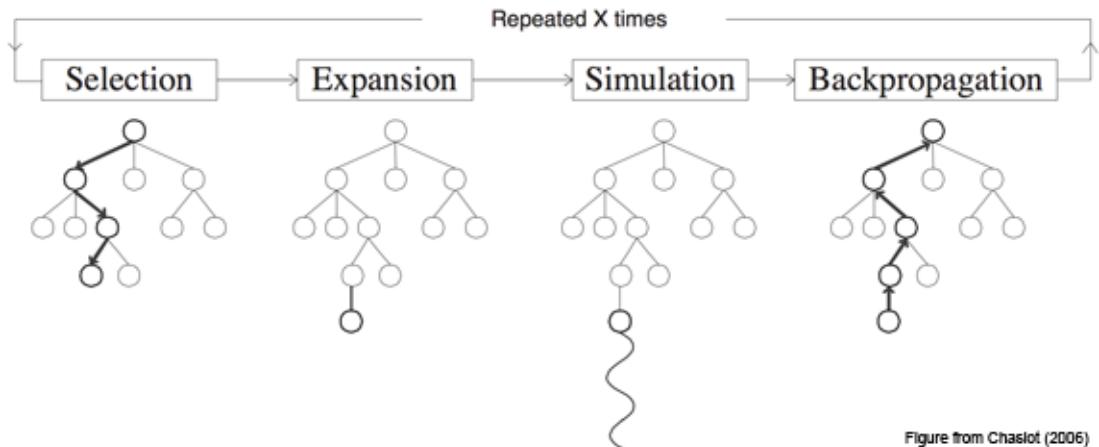


Figure from Chaslot (2006)

Ilustración 44 – 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%BAqueda_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 45 – 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.possible_movimientos
    fitnesses = calcular_fitnesses(movimientos)
    movimientos = movimientos.ordenar(fitnesses decrecientes)
    Para movimiento en tablero_actual.possible_movimientos:
        Para movimiento en movimientos:
            tablero_actual.simulacion(movimiento)
    ...

```

Tabla 46 – Cambios realizados en el pseudocódigo de alfa beta al incluir ordenación

Resultados algoritmos

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

Tabla 47 – Comparación de decisores de movimiento

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

Tabla 48 – 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

| <i>Concepto</i> | <i>Tiempo</i> | <i>Precio</i> | <i>Total</i> | <i>Acumulativo</i> |
|--|--|---------------|--------------|--------------------|
| <i>Elementos gráficos</i> | - | 25\$+25\$ | 50\$=44€ | 44€ |
| <i>Costes humanos</i> | 698 horas + 60 horas | 8€/hora | 6064€ | 6108€ |
| <i>Costes variables (electricidad)</i> | 768hr → PC 0.23KW 50hr → PC 0.045KW | 0.14€/KWH | 25€ | 6133€ |

Tabla 49 – Costes finales del proyecto

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 menos potencia es el usado durante las pruebas de la comunicación de red.

Los costes humanos han sido calculados según un sueldo medio de una persona sin experiencia, pero con formación universitaria.

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.

Elementos gráficos tienen un precio porque se compraron con sus correspondientes licencias en dos [Humble Bundles](#).

CONCLUSIONES

Este proyecto y su documentación hacen 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.

Como *opinión personal*, aunque el gasto de tiempo en este proyecto ha sido a todas luces excesivo, ya sea debido a la envergadura de los objetivos o a la inexperiencia del desarrollador, el resultado final es satisfactorio.

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.

Un saludo y hasta otra.

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/1BgkLwn66qN0oej3QEnjcF-4LunZpjKTA62uQIETTBw/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-ce1c05cfdf3>

<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/cs532I%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>

<SAVA DROW>

<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>