**TRABAJO FINAL DE GRADO**

**ALPHADSGAME: UN JUGADOR ARTIFICIAL INTELIGENTE**

Por: Enrique Albaladejo Herrero

DNI: 53716332Q

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Año 2023

Ingeniería informática

*A mis padres, sin vosotros no habría llegado tan lejos*

*Gracias por todo*

Índice de contenido

Índice de Imágenes

**No se encuentran elementos de tabla de ilustraciones.**

# Resumen

Este trabajo tiene como objetivo el desarrollo de un agente inteligente del juego ALPHADSGAME, así como una interfaz gráfica. Para ello, se ha decidido crear la parte gráfica del juego siguiendo la arquitectura cliente – servidor.

Para la parte del frontend he decidido utilizar [ReactJS](https://react.dev/), un framework de desarrollo web centrado en el desarrollo de interfaces por componentes. Gracias a esto, he conseguido modularizar distintos elementos de la interfaz, logrando mantener un código mucho más limpio y legible.

Por último, los datos se almacenarán en una base de datos de tipo relacional usando [MySQL](https://www.mysql.com/).

# Introducción

Dada la popularización de los videojuegos en la actualidad, estos se han convertido en un elemento fundamental de entretenimiento en la vida de muchos niños. Gracias al gran atractivo que estos generan, han ido surgiendo juegos que tienen como objetivo enseñar alguna habilidad. Un claro ejemplo es el “[Minecraft Education](https://education.minecraft.net/es-es)”, diseñado para ayudar a los estudiantes a explorar temas como historia, matemáticas y salud mental entre otros.

Con este mismo propósito, AlphaDSGame pretende familiarizar a los jugadores con el proceso de desarrollo de aplicaciones, los elementos necesarios para buscar el funcionamiento deseado de estas y el cumplimiento de los requisitos planteados al principio de su desarrollo.

## AlphaDSGame

A continuación, se detallará el funcionamiento del juego incluyendo sus reglas, elementos y las relaciones entre estos dentro de una partida.

AlphaDSGame es un juego de 4 jugadores donde cada uno tendrá que completar una misión con el objetivo de comunicar su circuito con el del siguiente jugador. Para ello, se disponen de una serie de elementos.

### Elementos

#### ­­Los componentes

Son las piezas que se colocan en el tablero con la finalidad de conectarse entre sí para formar un circuito que conecte con el tablero del siguiente jugador y así poder enviar información por el canal construido. Cada una de estas fichas tendrá un número de conexiones distintas que le permitirán conectarse con otras fichas. Además, al colocarlas se podrán rotar para que el jugado pueda elegir con qué posición colocarlas en el tablero.

Cabe destacar que cada componente tiene un valor de puntos asociado que variara en función del tipo, calidad y variante

Cada uno de estos componentes tiene un valor de puntos asociado que variara en función del tipo, calidad y variante. Los tipos en los que estarán clasificados son los normales y los especiales.

Los componentes normales son aquellos que pueden ser obtenidos por todos los jugadores. Varían según su número de conexiones y su material, este último indicará la calidad del componente.

* Componentes normales según número de conexiones: hay componentes de dos conexiones, de tres y de cuatro.

Imagen que contiene dibujo, señal

Descripción generada automáticamente

* Componentes según calidad: como se ha mencionado anteriormente, la calidad de un componente depende de su material. Los hay normales, de plata y de oro, calidad baja, media y alta respectivamente.

Icono

Descripción generada automáticamente

Los componentes especiales son aquellos que tienen el color de algún jugador, hay tres por cada jugador, tienen 4 conexiones, una en cada lado y tienen un título y descripción específicos. Habría que destacar que cada componente pertenece a un jugador y por lo tanto solo él podría comprar este componente y utilizarlo.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

#### El tablero

Todos los jugadores comenzaran la partida con un tablero propio en el cual habrá un solo componente inicial que viene por defecto. En el trascurso del juego, cada jugador tendrá que crear un circuito con los componentes que vaya consiguiendo en el mercado.

#### Los riesgos

Los riesgos son aquellos elementos peligrosos que aparecerán a lo largo de la partida en el mercado, se diferencian de los componentes porque son blancos, y bajo ellos aparece un botón de resolver, cuando aparece algún riesgo salta un modal que avisa al jugador de que debe resolver el riesgo lo antes posible, ya que, hasta que no se resuelven todos los riesgos el juego queda suspendido y ningún jugador podrá comprar ningún componente, el número de riesgos es limitado ya que hay 4 riesgos en el juego. Por último, cabe destacar que para resolver los riesgos se necesitan cartas de un subtipo específicas, las cuales se notificaran si mantenemos el cursor sobre el número de elementos necesarios para resolver este riesgo.

#### El mercado

El mercado es de donde salen los diversos componentes y riesgos que se necesitan comprar o resolver. Para ello se necesitarán unos elementos que hagan que se puedan comprar los diversos componentes, estos son las cartas. Bajo cada componente aparecerá una cajita con números, estos números representan la cantidad del tipo de cartas necesarias para poder comprar este componente especifico. En el caso del riesgo se indica el subtipo de cartas, así como la cantidad de ellas necesarias para poder resolver el riesgo. Cabe destacar que en ambos elementos aparece una sombra roja cuando no es posible comprar o resolver el objeto en cuestión.

#### Las cartas

Las cartas son los elementos que hacen posible la compra y resolución de los componentes y riesgos que existen en la aplicación. Hay tres tipos de cartas, las cartas de tipo “domain” (estas se dividen a su vez en “protected data”, “database”, y “team spirit”), tipo “computer science” (estas se dividen en “open source”, “antivirus” y “new technology”) y tipo “mathematics” (estas se dividen en “simple model”, “fast model” y “right model”). Cada vez que cada jugador haya jugado un turno se barajaran y repartirán cartas hasta que cada jugador se quede con 4.

Estos serían todos los elementos que forman parte del juego.

### Como se juega

En esta sección se explicará cómo se juega el juego que se va a implementar en físico.

Para comenzar son necesarios 4 jugadores, cada uno este encargado de una parte distinta en el desarrollo de la aplicación, estas partes se distinguen debido al color característico que tienen. Al comienzo de la partida, se reparten a cada jugador 4 cartas aleatoriamente y se ponen el resto encima de la mesa boca abajo. A continuación, se pone una tarjeta de mercado en el centro de la mesa y encima de los huecos se ponen 4 componentes que tienen costes distintos según la tarjeta, a continuación, si ha salido algún riesgo, se paraliza el juego y se intenta resolver el juego, mientras no se resuelva el juego, no se podrá continuar y lo único que podrán hacer los jugadores será descartarse e intercambiar cartas con el fin de solucionar el riesgo. Una vez se resuelve el riesgo se le da al jugador que lo resuelva unos puntos aleatorios. Si no hay riesgo, los jugadores tienen libertad para intercambiar cartas y comprar los componentes del mercado, no obstante, eso se deberá hacer en orden. Una vez comprados los componentes deberán ser ubicados en el circuito del jugador de forma que conecte con el resto de los componentes. Al terminar todos los jugadores su turno se volverá a repartir cartas y se colocaran nuevos componentes en el mercado hasta que haya 4 componentes normales. Además, la información que avanza por el circuito avanzará un componente en dirección al próximo jugador, es importante destacar que el jugador que disponga de este dato en su circuito deberá hacer lo que el grupo de jugadores decidan, perdiendo así la posibilidad de decidir qué hacer. Estas acciones deben de ser las que hagan que el dato salga lo antes posible del circuito del jugador.

# Propuesta

En primera instancia estuve barajando diversas tecnologías en las cuales implementar el juego, pero, tras un tiempo me decidí por implementar el juego siguiendo la estructura de cliente servidor en la cual hay dos procesos corriendo, por una parte, está el frontend que se encarga de la parte gráfica, la del cliente y por otra la parte del servidor que se encarga del procesado de datos, además el backend está conectado a una base de datos donde se guardarán los datos de cada juego.

## Tecnologías frontend

Para el frontend he decidido utilizar ReactJS ya que al investigar un poco vi que era una manera bastante simple y limpia de crear interfaces visuales, que además comparte bastantes similitudes con React Native, otro framework de desarrollo de interfaces en el cual se pueden exportar a distintos tipos de plataforma. No obstante, dado mi conocimiento de HTML, CSS y JS opte por utilizar ReactJS ya que muchas de las bases ya las conocía y me servían para crear la interfaz de una manera más cómoda respecto a React Native.

Pero antes de nada voy a introducir un poco React y cómo funciona.

### ReactJS

Tal y como he explicado antes React es un framework que te permite hacer de forma sencilla interfaces para web. Este framework es un framework para JS y es capaz de modularizar los distintos elementos de la web en Componentes que luego se podrán utilizar en distintas partes de la interfaz aprovechando tanto el código como los estilos y los elementos HTML.

#### El dom

Antes de explicar más en detalle cómo funcionan los componentes y como se relacionan entre sí, voy a explicar que es el dom y como hace react para mostrar las interfaces.

Cuando hablamos de dom nos estamos refiriendo al árbol de dependencias que existe entre un elemento html y su padre o sus hijos. Comúnmente, accedemos a este dom a través del objeto document. Con este objeto podemos añadir, eliminar y modificar etiquetas dentro de nuestra página web.

Una vez tenemos esto en mente, puedo comenzar a explicar cómo funciona react.

Al ejecutar el servidor de react se mostrará el contenido del index.html que se ubica en la carpeta public de nuestro proyecto, este contenido es muy escaso. No obstante, contiene lo justo y necesario para poder ejecutar nuestra aplicación.

Texto

Descripción generada automáticamente

A continuación, se ejecuta el index.js que contiene una instrucción clave en el renderizado de la interfaz. Esa instrucción de la que hablo es la llamada al método render de la librería “react-dom”, este método insertará el componente que tu le pases como primer parámetro en el elemento que tu le pases en el segundo parámetro, en este caso el div con id “root”.

Los desarrolladores somos capaces de aprovechar esta inserción para ir añadiendo lógica a nuestros componentes e ir construyendo interfaces complejas donde un componente contendrá a otros más pequeños.

Texto

Descripción generada automáticamente

En este ejemplo se aprecia como se inserta el componente “Provider” en el elemento con el id “root”, que a su vez contiene una serie de componentes que se muestran si la ruta de la página coincide con la estructura que se le indica en el path.

#### Componente

Un componente es un fragmento de la interfaz que se ha modularizado, siendo un ejemplo, una vista de la página web que estamos desarrollando. Estos componentes se pueden implementar de varias maneras, están los componentes de tipo clase y los componentes de tipo funcional, en esta aplicación he decidido utilizar componentes de tipo funcional. Estos se llaman a si debido a que básicamente son una función que recibe unos parámetros y devuelve un elemento que se insertará en el dom, y, por lo tanto, mostrándose por pantalla.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Este es un ejemplo de componente super básico en el cual se reciben unas propiedades como un objeto y devuelve un elemento HTML que contiene un div con el texto “holaMundo”.

Igual llegado a este punto te preguntas cual es la función de los props. La función de los props es comunicar el componente padre con el hijo. Supongamos que en vez de “holaMundo” quiero mostrar por pantalla el nombre que indique el padre del componente. El padre tendría que pasar la información al hijo y el hijo debería recibir y utilizar esa información. Esa comunicación se haría de la siguiente manera.

Texto

Descripción generada automáticamente

Como se aprecia en el ComponentePadre se llama al ComponenteHijo y se le pasa una propiedad name con el valor “Luis” de la misma manera que se haría con un elemento HTML, luego en el componenteHijo se puede acceder a esta propiedad a través de los props, lo único que en este caso me he servido de la desestructuración de JS para tener un código más legible, esto va a ser bastante recurrente.

No obstante, igual quieres hacer algo más dinámico, algo que cambie cuando pulses un botón, por ejemplo. Para ello, es necesario explicar que el componente se ejecuta cuando se renderiza y cuando cambian las props. Una vez renderizado, por mucho que se modifique el dato o constante, no se va a mostrar el cambio en la interfaz. Estos casos plantean un problema que es bastante crítico.

##### Estados

Por ello, se necesita un almacenamiento que, al modificarse, actualice el componente, este es el caso de los estados. En componentes funcionales, se crea un estado usando un hook, una función de react que utiliza el propio estado de react y permite hacer acciones especiales.

Texto

Descripción generada automáticamente

En este ejemplo se inicializa el estado usando el hook useState. Como valor por defecto tiene el string “Luis” que se pasa como prop al hijo. Al pulsar el botón, se cambia el estado y renderiza de nuevo el componente pasando un nuevo prop a ComponenteHijo que mostrará “Alberto”.

No obstante, los estados no están pensados como una memoria a largo plazo y por lo tanto es necesario utilizar otro tipo de memoria, el store, por ejemplo.

##### Store

Un store es una memoria de larga duración, que se mantiene siempre que el componente Provider este renderizado. No obstante, en el Provider tienes que pasar una prop store que se obtiene ejecutando esta función.

Texto

Descripción generada automáticamente

A esta Store se puede acceder a través de la función useSelector o de la función connect que recibe como parámetros una función que maneja el estado y permite meter como prop los datos obtenidos del estado y otra función que recibe una variable dispatch, la cual, sirve para hacer modificaciones al store desde cualquier componente.

Estas modificaciones se configuran en un reducer que es una función que recibe un estado y una acción y retorna el estado modificado. Dentro de este action, utilizo la variable type para determinar qué acción se va a ejecutar y paso la información necesaria para realizar la acción.

Un detalle importante es comentar que utilizo la función combineReducer de redux para fusionar ambos reducers y separar un poco la lógica de cada parte de la aplicación.

## Tecnologías backEnd

Como tecnologías de back he decidido usar Flask, una librería de Python orientada a desarrollo de servidores.

### Flask

Flask es una librería de Python orientada a servidores que he utilizado para crear un api rest.

Texto

Descripción generada automáticamente con confianza media

Este es el código que inicializa el servidor de Flask. Tras este código se declaran los métodos los cuales se declaran de la siguiente manera.

Texto

Descripción generada automáticamente

Se utiliza la notación @app.route para indicar la url y el tipo del endpoint que va a relacionarse con la función. Además, esta función devolverá un json con los datos de respuesta del endpoint.

## Tecnologías adicionales

Además de ReactJS y Flask he utilizado otras tecnologías tales como MySQL y GitHub.

### MySQL

MySQL es una tecnología de bases de datos relacionales, en mi caso utilizo esta tecnología para guardar la información de los distintos juegos.

Texto

Descripción generada automáticamente

Para hacer más sencilla la configuración de la base de datos decidí crear un fichero de código que ayude a crear la base de datos

### GitHub

## Proceso de desarrollo

### Frontend

#### Flujos

### Backend

#### Lógica de aplicación

#### Lógica de agente

### Etapas