**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 contenidos

[Resumen 5](#_Toc151306985)

[Introducción 7](#_Toc151306986)

[AlphaDSGame 7](#_Toc151306987)

[Elementos 7](#_Toc151306988)

[Los componentes 7](#_Toc151306989)

[Las cartas 8](#_Toc151306990)

[Los riesgos 9](#_Toc151306991)

[El mercado 9](#_Toc151306992)

[El tablero 9](#_Toc151306993)

[Como se juega 9](#_Toc151306994)

[Propuesta 11](#_Toc151306995)

[Proceso de desarrollo 11](#_Toc151306996)

[Tecnologías 11](#_Toc151306997)

[MySQL 11](#_Toc151306998)

[GitHub 12](#_Toc151306999)

[Frontend 12](#_Toc151307000)

[Tecnologías 12](#_Toc151307001)

[ReactJS 12](#_Toc151307002)

[El dom 13](#_Toc151307003)

[Componente 14](#_Toc151307004)

[Estados 15](#_Toc151307005)

[Store 16](#_Toc151307006)

[Versiones 17](#_Toc151307007)

[Versión 0.1 (47fd8ca): 17](#_Toc151307008)

[Flujos 18](#_Toc151307009)

[Comenzar partida 18](#_Toc151307010)

[Comprar componentes 18](#_Toc151307011)

[Intercambiar cartas 20](#_Toc151307012)

[Versión 0.2 (f206803) 20](#_Toc151307013)

[Flujos 20](#_Toc151307014)

[Comenzar Partida 21](#_Toc151307015)

[Comprar componentes 21](#_Toc151307016)

[Intercambiar cartas 22](#_Toc151307017)

[Versión final 1.0 (c99806d) 23](#_Toc151307018)

[Flujos 23](#_Toc151307019)

[Comenzar partida 23](#_Toc151307020)

[Comprar componentes 24](#_Toc151307021)

[Intercambiar cartas 24](#_Toc151307022)

[Librerías 25](#_Toc151307023)

[Backend 25](#_Toc151307024)

[Tecnologías 25](#_Toc151307025)

[Flask 25](#_Toc151307026)

[Lógica de aplicación 25](#_Toc151307027)

[Lógica de agente 25](#_Toc151307028)

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

Todos componentes especiales tienen 4 salidas, se dividen en función del número de jugadores y se clasificaran según el color del jugador correspondiente. Siguiendo esta dinámica, cada jugador sólo tendrá tres componentes especiales que podrá comprar siempre y cuando pertenezcan a su color.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

#### Las cartas

Las cartas se reparten a cada jugador al principio de la partida. Son el medio que hace posible la compra de componentes y la resolución de los riesgos (estos últimos serán explicados a continuación).

Hay tres tipos, domain, computer sicience y mathematics, cada uno de los cuales a su vez agrupan 3 subtipos:

* Domain
  + Protected data
  + Database
  + Team spirit
* Computer science:
  + Open source
  + Antivirus
  + New tecnology
* Mathematics:
  + Simple model
  + Fast model
  + Right model

Cada vez que cada jugador haya jugado un turno se barajaran y repartirán cartas hasta que cada jugador se quede con 4.

#### Los riesgos

Los riesgos son aquellos elementos peligrosos que aparecerán en el mercado a lo largo de la partida, se diferencian de los componentes porque son blancos y tienen un botón de resolver bajo ellos.

Al aparecer un riesgo, aparecerá un modal que avisará al jugador de que debe resolver el riesgo lo antes posible. Hasta que el jugador no resuelva todos los riesgos el juego queda suspendido y ningún jugador podrá comprar ningún componente.

A pesar de que hay varios tipos de riesgo, el número de apariciones de estos se limita a 4 por partida. Los tipos de riesgo están asociados a los subtipos de las cartas por lo que un riesgo solo se podrá solucionar con una o dos cartas del subtipo que le corresponde.

#### El mercado

El mercado es el lugar donde los jugadores podrán comprar componentes y solucionar riesgos mediante el uso de cartas. Estos riesgos y componentes aparecerán en el mercado junto con un grupo de números que representan la cantidad del tipo de cartas necesarias para realizar una compra o una resolución.

* Venta de cartas:
* Resolución de riesgos:

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.

#### 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.

### 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

A continuación, se presenta mi propuesta de tfg.

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 parte el 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.

## Proceso de desarrollo

Este apartado pretende describir el proceso y el orden que se ha seguido a lo largo del desarrollo del código.

El primer paso del desarrollo fue investigar acerca del juego con la intención de comprender su funcionamiento. Tras esto, me di cuenta de algunas inconsistencias dentro de la dinámica del juego.

Lo primero que decidí cambiar fue la forma en la que los jugadores consiguen los puntos que determinan el ganador del juego.

En un inicio, tras cada turno se repartían unos puntos sin tener en cuenta las acciones que había realizado cada jugador. Además, se puntuaba un extra por la longitud máxima del camino que se había creado y por el número de componentes colocados.

Tras pensarlo detenidamente, decidí cambiar este sistema de puntuación estableciendo una serie de acciones que, a pesar de que previamente estaban compensadas y se repartían puntos extra por realizarlas, ahora las acciones difíciles se premian aleatoriamente dentro de un rango mayor al resto. Con ello implemento una forma de puntuación más intuitiva para el usuario.

A continuación, decidí las tecnologías y el flujo de trabajo que usaría para el desarrollo del videojuego.

### Tecnologías

Desde el principio el desarrollo ha seguido el flujo de trabajo gitflow. Este consiste en tener dos ramas de trabajo, una de desarrollo y una para la versión estable, en mi caso la rama main es la estable y la dev la de desarrollo.

#### 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 ayudé a crearla junto con las tablas necesarias para el correcto funcionamiento del juego.

#### GitHub

GitHub es una plataforma utilizada para la gestión de versiones de aplicaciones con git.

Gracias a ello me ha permitido mantener un proceso de desarrollo paralelo del mismo código en dos ordenadores, además de que me permite mantener una versión estable y una en desarrollo.

Una vez tuve las tecnologías decididas ya solo faltaba comenzar con el código.

## Frontend

En cuanto al frontend, este ha pasado por varios cambios, los cuales he podido mantener un registro al utilizar git, en total he tenido tres versiones más o menos estables en el frontend:

### Tecnologías

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 tú le pases como primer parámetro en el elemento que tú 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.

### Versiones

Voy a explicar un poco el histórico de versiones. Como tal diferencio tres versiones estables del juego. A medida que avancen las versiones se vera como se han ido solucionando errores que he ido teniendo y que ya están solucionados en la versión final.

#### Versión 0.1 (47fd8ca):

Esta es la primera versión funcional del juego, esta versión se caracteriza por la ausencia de un backend. Por ello, implemente toda la interfaz sirviéndome solo de las herramientas del front, es decir, la lógica se encontraba en el store. Obviamente, esto no es correcto y necesitaba trasladar esa lógica al back, cosa que posteriormente hice una vez acabé la primera versión funcional del juego.

##### Flujos

A continuación, mostrare los distintos flujos que tiene esta versión, no obstante, no mostraré nada del código hasta la versión final ya que no quiero profundizar demasiado en estas versiones, solo quiero mostrar los cambios sucedidos a lo largo del desarrollo en cuanto a la interacción con el usuario.

###### Comenzar partida

Gráfico en cascada

Descripción generada automáticamente con confianza media

En un comienzo esta vista tenía la siguiente interfaz, la página estaba completamente en blanco, y, en una tarjeta centrada se encontraba el título, los botones de cada jugador y el botón de comenzar. Para cambiar el tipo de jugador se pulsaba en el recuadro de color repetitivas veces pudiendo elegir así entre jugador, bot o ninguno, el jugador, además, tiene la posibilidad de cambiar su nombre por el que desee.

###### Comprar componentes

Interfaz de usuario gráfica, Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

Este es el resultado de la primera iteración que le hice a este flujo, como se puede ver, la interfaz se divide en dos partes, la parte de la izquierda, que contiene el mercado, las cartas y la información de los jugadores así como una serie de botones para intercambiar cartas y finalizar el turno, y la parte de la derecha, donde se muestra el tablero del jugador junto con su misión asignada, esta misión al cerrar el circuito cambiará y mostrará otro mensaje que dependerá del color del jugador.

Centrándonos mas en el mercado, este contiene los componentes y los riesgos junto a sus costes, que además indicaran con una sombra roja si pueden o no interactuar con ellos. Los componentes normales tienen la posibilidad de rotarse, pero esta rotación no tiene ningún feedback, con esto me refiero a que no se le da a entender al usuario en ningún momento que tiene la posibilidad de rotar el componente.

Por último, para comprar el componente hay que arrastrarlo hasta la casilla en la que se desee colocar, este componente no se podrá colocar si no se dispone de las cartas seleccionadas suficientes, cabe mencionar que al arrastrarlo se utiliza el drag por defecto de cuando arrastras una imagen.

Interfaz de usuario gráfica, Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

Así quedaría tras haber colocado el componente, como se puede apreciar, el componente se muestra con un fondo verde claro, el mismo que contiene el borde del tablero.

###### Intercambiar cartas

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteComo se puede apreciar, este flujo contiene un modal que, a su vez, contiene los jugadores del juego, el jugador, tendrá que seleccionar las cartas a intercambiar, tanto las suyas como las otras, una vez hecho esto, el jugador que propone el intercambio pulsa el botón de aceptar y el jugador al que se le propone el intercambio tendrá que pulsar el otro botón en el que pone aceptar. Esta iteración no es la más optima si pensamos en un juego en línea, no obstante, me base en la idea de que todos los jugadores estarán jugando desde el mismo ordenador para desarrollar esta interfaz, sin embargo, entiendo que orientar esto de esta manera es muy peligroso y no queda del todo claro, por lo que a futuro decidí modificar casi la totalidad del flujo en cuestión.

#### Versión 0.2 (f206803)

##### Flujos

En esta sección voy a relatar los distintos flujos que hay en la aplicación y el cambio que ha ido teniendo el estilo a lo largo de las versiones.

###### Comenzar Partida

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

En una segunda revisión de esta interfaz hice que los campos del nombre se vieran siempre y cambie la interacción con la cual el usuario seleccionaba que tipo de jugador era, ahora al pasar el cursor por encima saldrían unas flechas que permiten cambiar el tipo.

###### Comprar componentes

En este flujo se colocan componentes, para ello se deberán seleccionar las cartas necesarias, y, posteriormente, arrastrar el componente hasta la casilla del tablero del jugador. Cabe destacar que se da la posibilidad al usuario de rotar el componente.

Interfaz de usuario gráfica, Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

En esta versión, se comienza a utilizar la librería *react-beautiful-dnd* que añade la posibilidad de tener un drag and drop bastante bonito respecto a cómo estaba antes.

Icono

Descripción generada automáticamente con confianza mediaSe añaden además un par de botones encima de los componentes que dan la posibilidad de rotarlos.

###### Intercambiar cartas

Interfaz de usuario gráfica

Descripción generada automáticamente

En esta versión se mantiene el flujo, no sufre ningún tipo de cambios.

#### Versión final 1.0 (c99806d)

Mas tarde me pareció correcto cambiar la lógica que había detrás del juego, en esta nueva lógica la interfaz usaría el id para acceder a un juego ya existente mediante la ruta.

Además, decidí mantener inglés y español como idiomas soportados, iniciándose el juego en el idioma del navegador, con esto soluciono los problemas con los idiomas existentes y añado un poco de internacionalización al juego.

También añadí una sección de instrucciones para aquellas personas que tengan alguna duda sobre cómo funciona la aplicación a la que se puede acceder desde un botón en la cabecera.

##### Flujos

###### Comenzar partida

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Por último, utilice el formato que he empleado en el resto del juego y añadí la misión principal del mismo, para tener un poco más de contexto sobre de que va el juego.

###### Comprar componentes

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

Esta es la versión final del juego, es la que ha sufrido más cambios, prácticamente cualquier elemento en la interfaz ha sufrido algún cambio con el objetivo de tener una interfaz más estética.

Hablando de los componentes, se ha hecho de 0 un componente encargado de renderizar un svg correspondiente a cada componente, esto ayuda a mejorar la calidad de estos puesto que antes el componente era una imagen a una calidad determinada.

En cuanto al mercado, sus esquinas están redondeadas, está más centrado en la pantalla y el logo del juego está ubicado en la esquina inferior derecha. Nótese también, que los componentes de dentro ahora están ligeramente redondeados y sus costes se encuentran en una caja con el color del mercado. Todos estos cambios ayudan a tener una interfaz mucho más minimalista que a mi parecer tiene un mejor acabado que antes.

Por otra parte, las cartas han cambiado de tamaño, y ahora se colocan centradas con respecto a la parte izquierda de la interfaz.

En cuanto al tablero, también sufrió cambios, se han añadido conexiones en el lateral izquierdo y animaciones en el cambio de jugador para hacer entender que cada jugador conecta con el siguiente, además, se ha cambiado la disposición del texto en la parte de las misiones.

Además, nótese que el color y la misión de los jugadores en esta versión ya está arreglado.

###### Intercambiar cartas

Este flujo consiste en poder seleccionar un jugador e intercambiar cartas con él.

#### Librerías

## Backend

### Tecnologías

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.

lTexto

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.

### Lógica de aplicación

Diagrama, Esquemático

Descripción generada automáticamente

### Lógica de agente