ADVENTURE GAME

ARQUITECTURA DO SOFTWARE

Juan Falcón Abal
Juan Filgueiras Rilo
Marco Martínez Sánchez
Pablo Regueiro Teijido

Julia Álvarez Gurdiel
Jorge Casal Munin
Miguel Cortés Cambeses
Ishwara Coello Escobar

UNIVERSIDADE DA CORUÑA

fic
facultade de informática da coruña

ÍNDICE

1. INTRODUCCIÓN

Nuestra práctica consiste en un juego ejecutable únicamente mediante terminal basado en combates 1 contra 1.

La práctica consiste en la realización de un juego ejecutable por línea de comandos basado en combates uno contra uno entre los diferentes usuarios que estén adscritos al juego.

Para la implementación de este juego se han discutido las diferentes arquitecturas distribuidas estudiadas en la asignatura, las cuales tienen cada una sus ventajas y sus inconvenientes, y por lo cual se ha tomado una decisión rigurosa a la hora de escoger la arquitectura más adecuada para el juego escogido.

Como documentación gráfica adicional, se han realizado las diferentes diagramas en la representación C4 tal y como se ha estudiado en la asignatura, distinguiendo entre los diagramas de contexto, contenedores y componentes, que darán un mayor detalle de todos los módulos implementados en el juego.

Además, se dispone de un manual de usuario que contiene todas las opciones a realizar por parte de los usuarios que ejecuten el juego. Todos estos apartados serán mencionados en los siguientes apartados con mayor detalle.

2. ARQUITECTURA ELEGIDA

Se ha escogido una arquitectura peer-to-peer. Es una arquitectura participativa, con compartición de recursos entre pares. Esta arquitectura posee una gran capacidad para la escalabilidad y la disponibilidad, gracias al reparto uniforme y descentralizado de la carga.

Es un sistema descentralizado debido a que cada *peer* puede atender a una petición recibida.

Conexión full mesh: Tipo de estructura en la que todos los peer están conectados entre ellos, pero no es lo normal.

3.TÁCTICAS EMPLEADAS

3.1 TÁCTICAS DE RENDIMIENTO

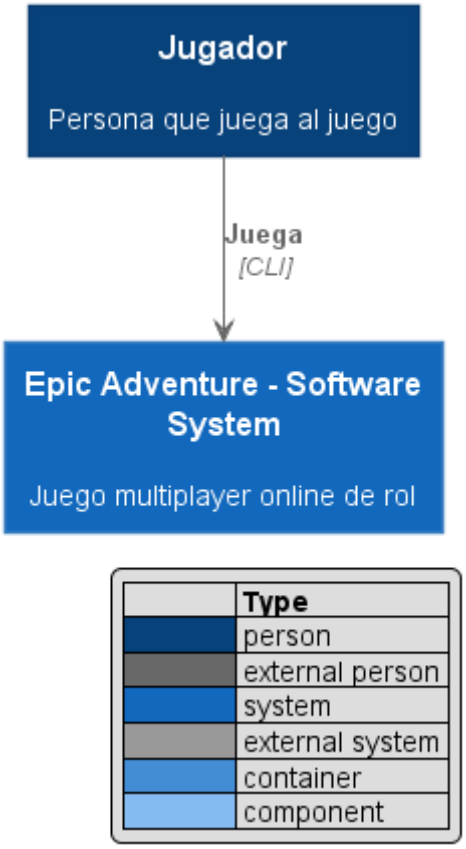
- **Introducción de concurrencia:** Esta táctica, empleada para la gestión de recursos, se basa en la ejecución de cada uno de los módulos en un proceso distinto.
- **Limitación del tamaño de las colas:** Aplicamos esta táctica de rendimiento, que se emplea para la demanda de recursos. En nuestra práctica lo que hacemos es limitar la cantidad de eventos pendientes, de esta forma si la cola está llena y nos llegan más peers estos serán descartados. Con esto intentamos que el rendimiento global del sistema sea satisfactoria.

3.2 TÁCTICAS DE DISPONIBILIDAD

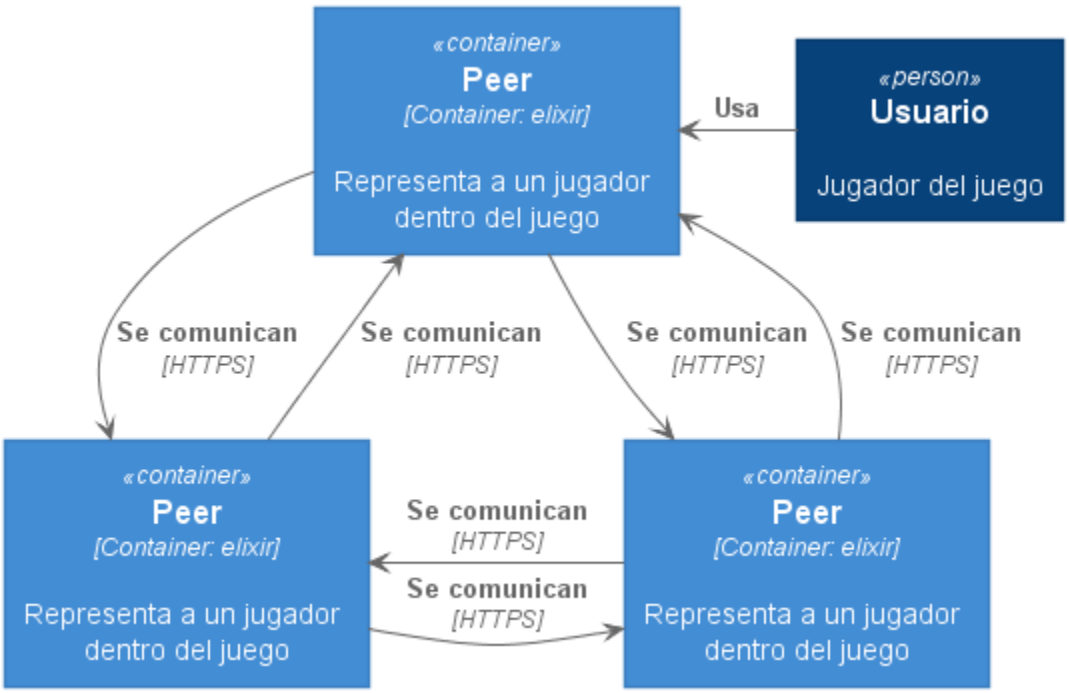
- **Ping/echo:** En esta táctica, utilizada para detección de errores, un peer realizará un ping cada 2 segundos para comprobar la disponibilidad del resto de componentes. El emisor esperará durante un tiempo preestablecido (en este caso 3 segundos) para recibir un echo.
- **Monitor de procesos:** Esta táctica es empleada para la prevención de errores. En nuestro caso, los peers controlan a sus peers conocidos, para así poder detectar algún problema de funcionamiento en esos peers.

4. DIAGRAMAS C4

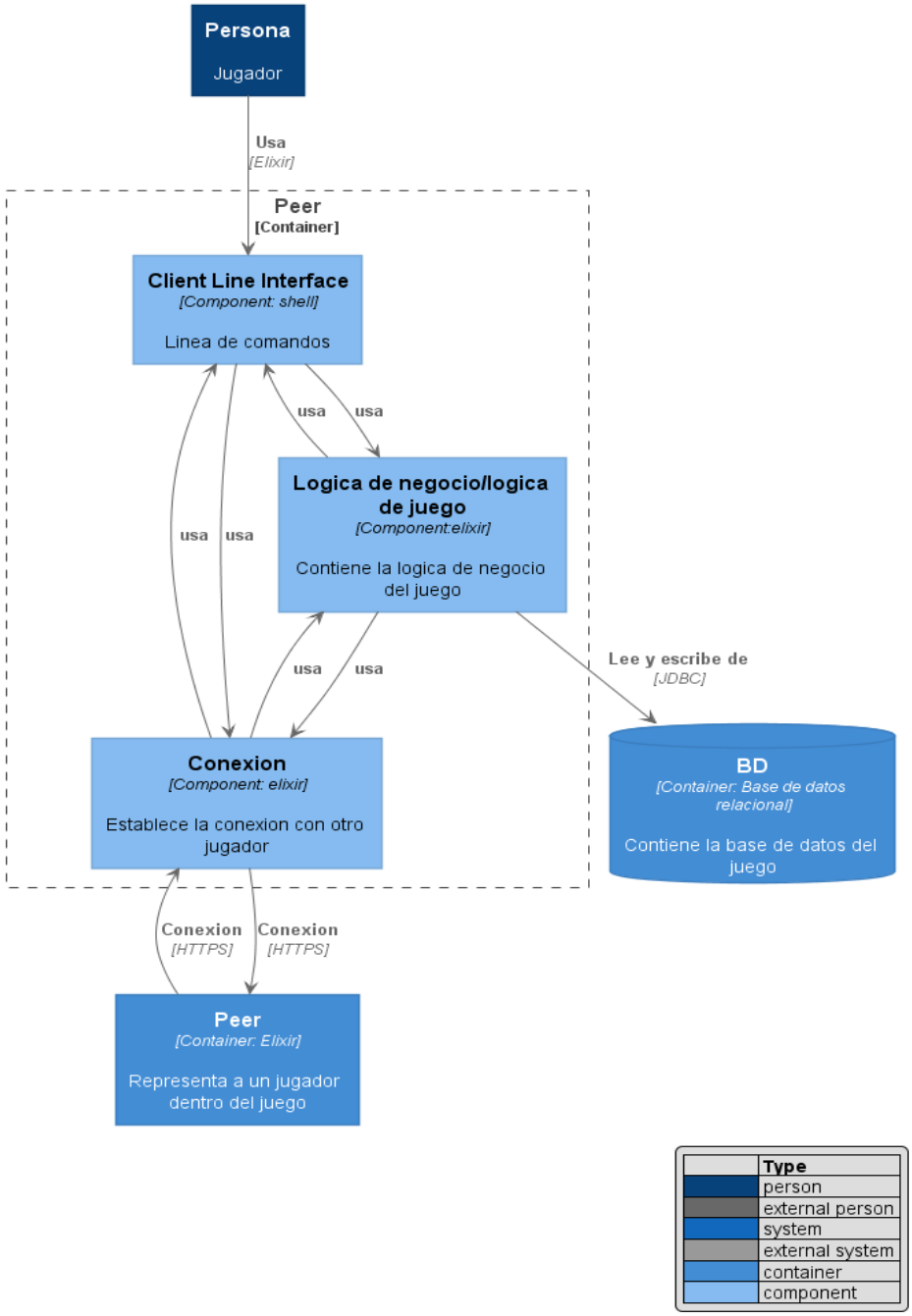
4.1 DIAGRAMA DE CONTEXTO



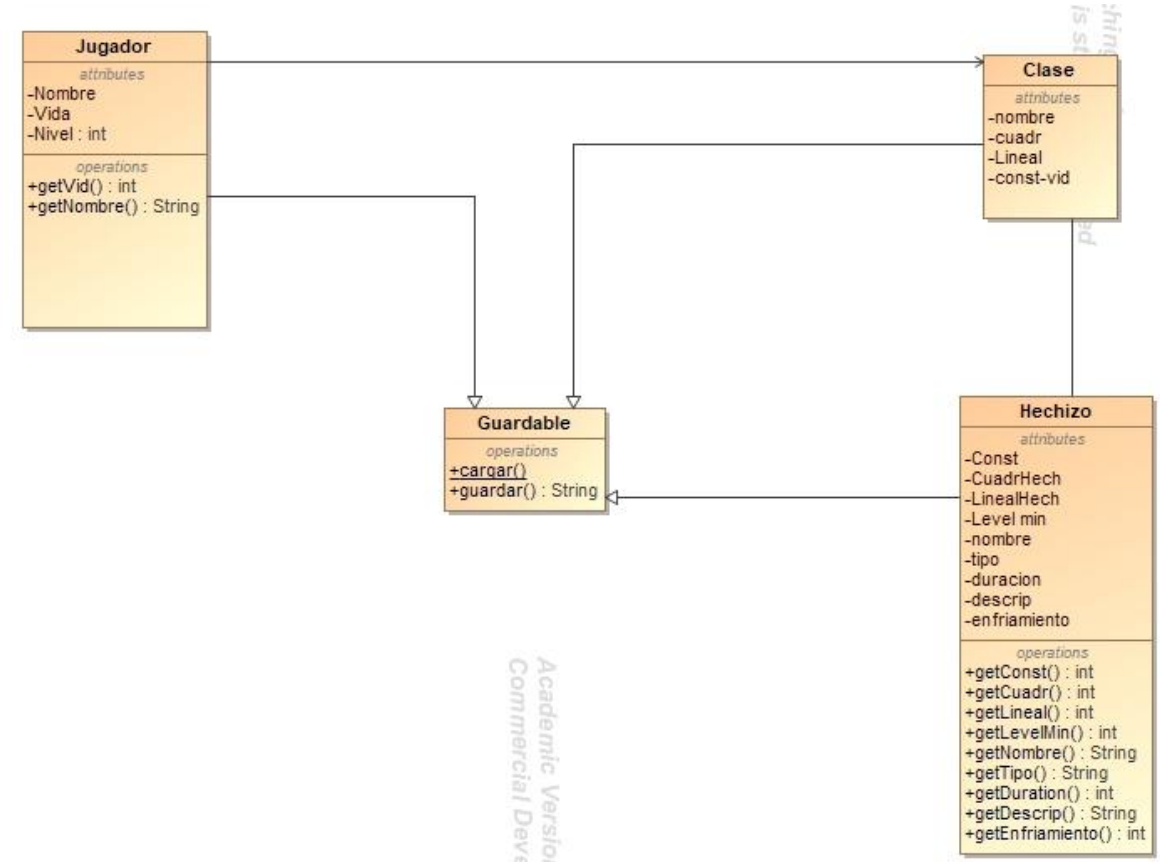
4.2. DIAGRAMA DE CONTENEDOR



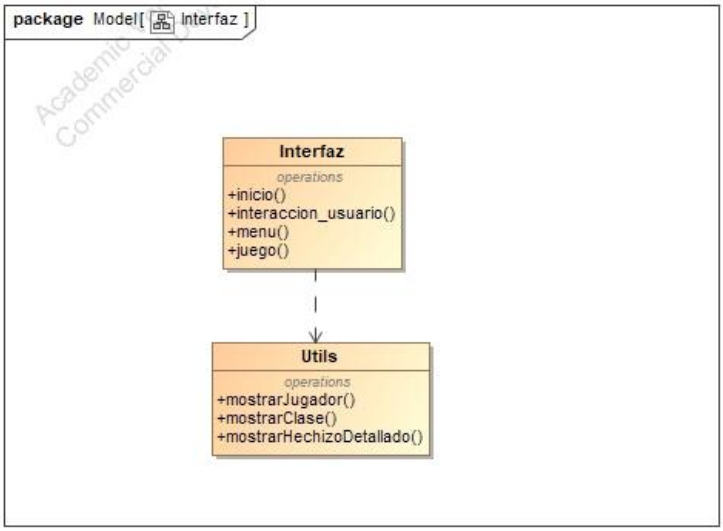
4.4. DIAGRAMA DE COMPONENTE (PEER)



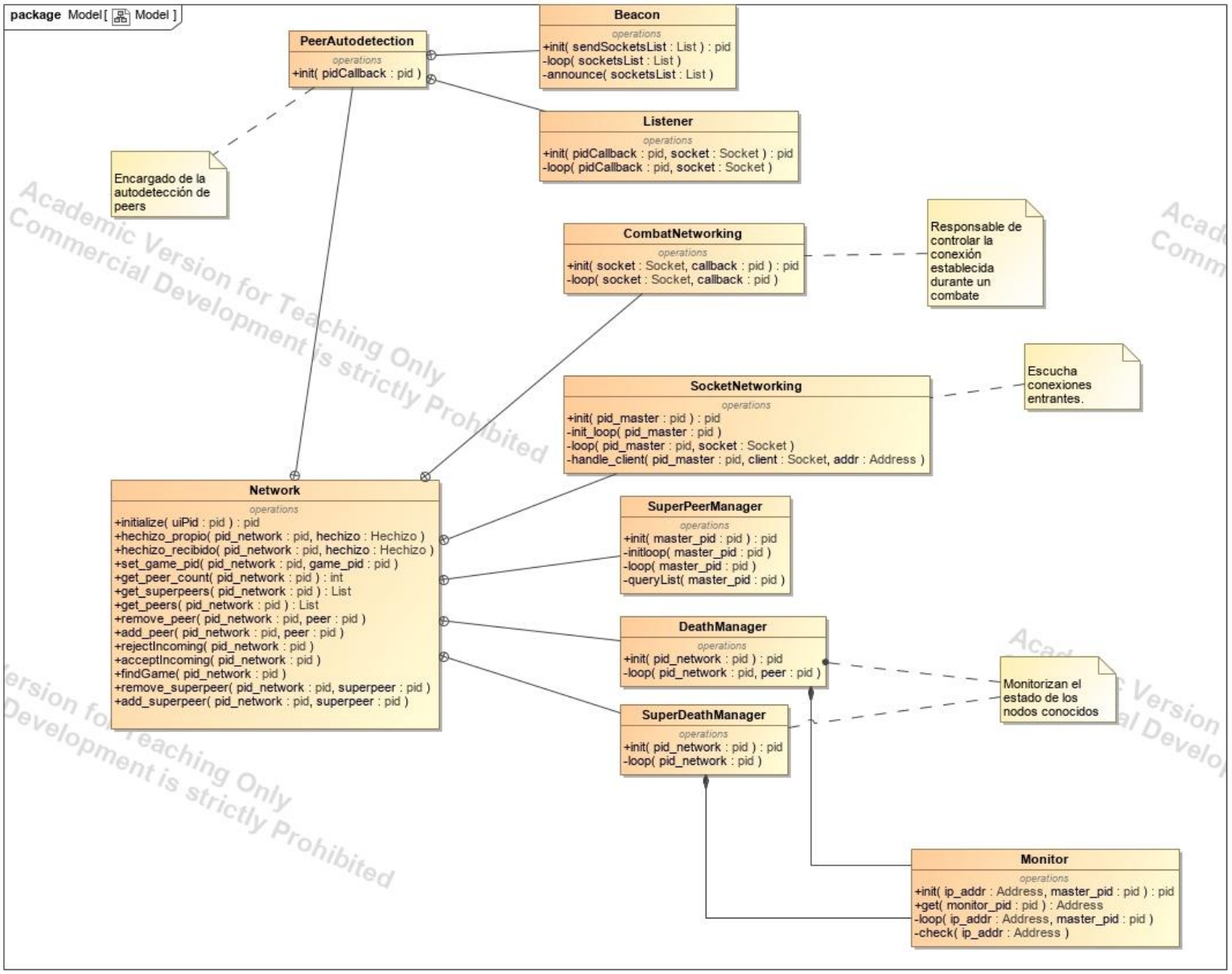
UML DE LÓGICA DE JUEGO



UML DE INTERFAZ



UML DE CONEXIÓN PEER



5. MANUAL DE USUARIO

INICIALIZACIÓN

iniciar(nombreArchivo, pidCallback)

- **Resumen:** Arranca la lógica de juego, cargando la información de las clases y sus hechizos del fichero especificado. No se podrá empezar a jugar hasta que, o bien se carguen los datos del jugador con el método "cargar", o bien se cree uno nuevo con el método "crearJugador".
- **Argumentos:**
    - **nombreArchivo:** El nombre del archivo que se usará para cargar la información de las clases.
    - **pidCallback:** El pid de la interfaz que se usará para comunicar eventos (actualmente no implementado).
- **Valor de retorno:**
    - **{:ok, pid}** El átomo ok, junto con el pid del proceso de control del estado del juego.
- **Errores:**
    - **:error** Ocurrió un error durante la carga de los datos iniciales.

CREARJUGADOR

- **Resumen:** Crea un nuevo jugador de nivel 1 para poder empezar a jugar.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
    - **jugador:** Los datos del jugador que deben haber sido creados llamando a la función Jugador.constructor/3
- **Valor de retorno:**
    - **:ok** No hubo ningún error durante la creación.
- **Errores:**
    - **:estadoInválido** Los datos del jugador ya habían sido cargados.

ENTRADA/SALIDA DE ARCHIVOS

GUARDAR

- **Resumen:** Guarda los datos del jugador en el archivo especificado.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
    - **fileNamePlayer:** El nombre del archivo donde se guardarán los datos del jugador.
- **Valor de retorno:**
    - **:ok** No hubo ningún error durante el guardado.
- **Errores:**
    - **:estadoInválido** Todavía no se tienen los datos del jugador cargados.
    - **:error** Hubo un error y el guardado no pudo llevarse a cabo.

CARGAR

- **Resumen:** Carga los datos del jugador del archivo especificado.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
    - **fileNamePlayer:** El nombre del archivo de donde se cargarán los datos del jugador.
- **Valor de retorno:**
    - **:ok** No hubo ningún error durante la carga.
- **Errores:**
    - **:estadoInválido** Los datos del jugador ya habían sido cargados.
    - **:error** Hubo un error y la carga no se pudo llevar a cabo.

OBTENCIÓN DE INFORMACIÓN

LISTARCLASES

- **Resumen:** Devuelve una lista conteniendo todas las clases cargadas por el juego.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
- **Valor de retorno:**
    - **clases** La lista de clases

OBTENERCLASE

- **Resumen:** Busca los datos de una clase concreta a partir de su nombre.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
    - **clase:** El nombre de la clase a buscar.
- **Valor de retorno:**
    - **clase** Los datos de la clase.
- **Errores:**
    - **:notFound** No se encontró ninguna clase con el nombre especificado.

OBTENERJUGADOR

- **Resumen:** Obtiene los datos del jugador.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
- **Valor de retorno:**
    - **jugador** Los datos del jugador.
- **Errores:**
    - **:estadoInválido** Los datos del jugador no se habían cargado todavía.

COMBATE

SYNCOMBATE

- **Resumen:** Inicia el proceso de establecimiento de combate con otro jugador. Devuelve el pid y los datos del jugador, y prepara el estado interno para el combate. Para establecer un combate, es necesario llamar a este método en uno de los dos peers, pero no en los dos.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
- **Valor de retorno:**
    - **{pid, jugador}** Par conteniendo el pid del proceso de control del estado del juego, y los datos del jugador.
- **Errores:**
    - **:estadoInválido** Ya estaba en combate, o bien los datos del jugador no fueron cargados todavía.

ACKCOMBATE

- **Resumen:** Continúa el proceso de establecimiento de combate con otro jugador. Recibe el pid del proceso de control del juego del otro jugador (o el de un módulo de red que actúe como repetidor), y los datos de su jugador. Devuelve el pid y los datos de este jugador, e inicia el combate en este peer. Para establecer un combate, es necesario llamar a este método en ambos peers, después de haber ejecutado el proceso de syn en uno de ellos.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
    - **pidRed:** El pid del otro proceso de control del estado del juego, o de un componente que actúe como repetidor entre ambos. El control del estado enviará mensajes para mantener la sincronización a este pid (no implementado todavía).
    - **datosEnemigo:** Los datos del jugador enemigo.
- **Valor de retorno:**

- **{pid, jugador}** Par conteniendo el pid del proceso de control del estado del juego, y los datos del jugador.

- **Errores:**
  - **:estadoInválido** Ya estaba en combate, o bien los datos del jugador no fueron cargados todavía.

RETIRARSE

- **Resumen:** Te retiras del combate, manteniendo la vida que tenías en ese momento para el próximo combate.
- **Argumentos:**
  - **pid:** El pid del proceso de control del estado del juego.
- **Valor de retorno:**
  - **:ok** Te retiraste del combate.
- **Errores:**
  - **:estadoInválido** No estabas en combate.

USARHECHIZOPROPIO

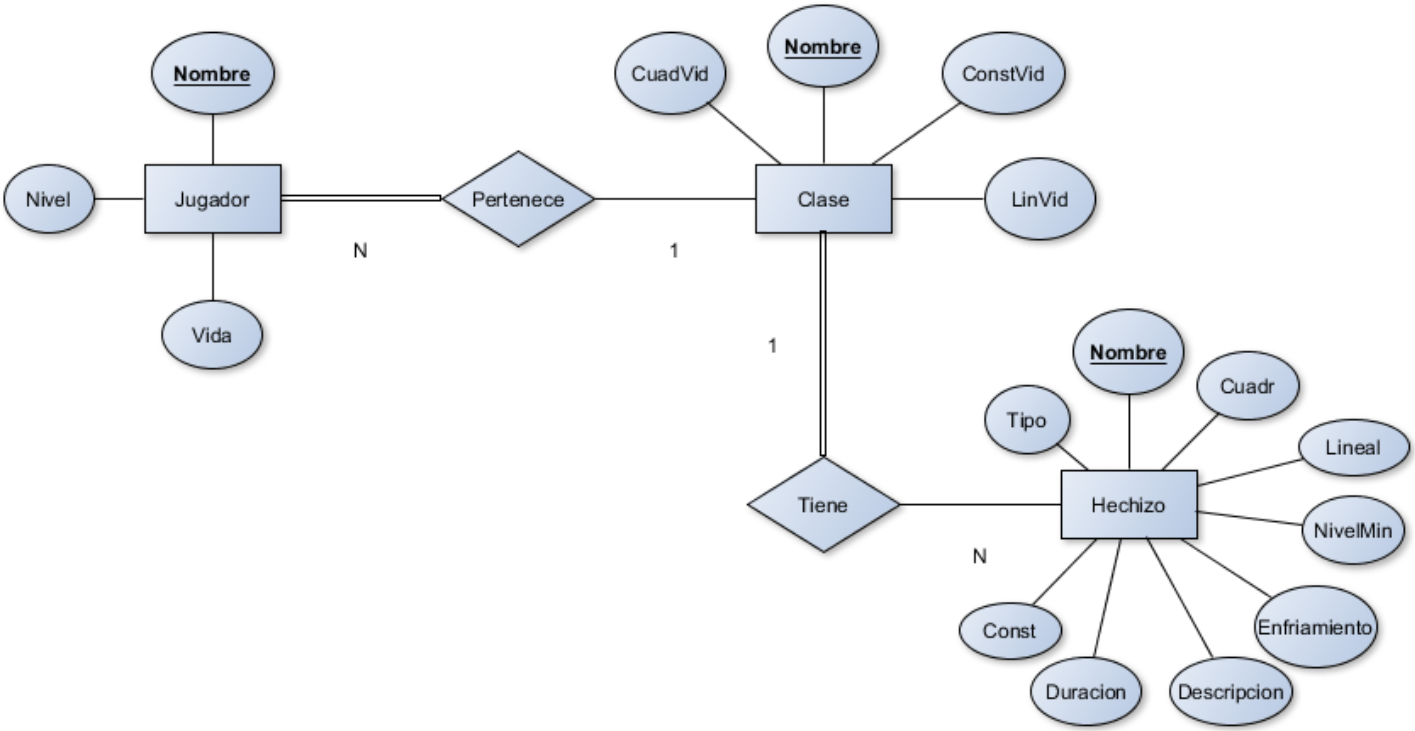
- **Resumen:** Utilizas un hechizo con tu personaje durante el combate. Ejecutar el hechizo implica que se avance un turno tuyo en la simulación.
- **Argumentos:**
  - **pid:** El pid del proceso de control del estado del juego.
  - **hechizo:** Los datos del hechizo a utilizar.
- **Valor de retorno:**
  - **:continuar** El hechizo se pudo lanzar correctamente, y el combate sigue adelante.
  - **:victoria** El hechizo se pudo lanzar correctamente, y se ganó el combate.
- **Errores:**
  - **:estadoInválido** No estabas en combate.
  - **:turnoInválido** No era tu turno

UsarHechizoRemoto

- **Resumen:** El enemigo utiliza un hechizo con su personaje durante el combate. Ejecutar el hechizo implica que se avance un turno suyo en la simulación.
- **Argumentos:**
    - **pid:** El pid del proceso de control del estado del juego.
    - **hechizo:** Los datos del hechizo a utilizar.
- **Valor de retorno:**
    - **:continuar** El hechizo se pudo lanzar correctamente, y el combate sigue adelante.
    - **:derrota** El hechizo se pudo lanzar correctamente, y se perdió el combate.
- **Errores:**
    - **:estadoInvalido** No estabas en combate.
    - **:turnoInvalido** No era el turno del enemigo.

6. DOCUMENTACIÓN ASOCIADA

DIAGRAMA ENTIDAD/RELACIÓN DE LÓGICA DE JUEGO



FACHADA DE LÓGICA DE JUEGO

Esta clase actúa como punto de entrada en el componente de lógica de juego, permitiendo ejecutar todas las operaciones necesarias para controlarlo y obtener información de él.

EJEMPLO DE USO

- **Peer 1:**

```
# Se cargan los datos del juego
{:ok, pid1} = GameFacade.iniciar("archivo", self())

# Se cargan los datos del jugador
GameFacade.cargar(pid1, "archivo2")

# Se empieza a establecer un combate
{pid1, jugador1} = GameFacade.synCombate(pid1)
```

- **Peer 2:**

```
# Se cargan los datos del juego
{:ok, pid2} = GameFacade.iniciar("archivo", self())

# Se cargan los datos del jugador
GameFacade.cargar(pid2, "archivo2")

# Se inicia el combate tomando los datos del peer1
{pid2, jugador2} = GameFacade.ackCombate(pid2, pid1, jugador1)
```

- **Peer 1:**

```
# Se inicia el combate tomando los datos del peer2
GameFacade.ackCombate(pid1, pid2, jugador2) # Aquí el combate comienza en ambos lados
```