Distribución de Clases:

- Jugador -Nico
- Partida - Aleix
- EntradaRanking - Nico
- Ranking - Nico
- Hidato - Eloi
- InterficieUsuario - Eloi
- Celda - Nico
- Graph - Eloi y Aleix
- Node - Eloi

Funcionalidades Primera Entrega:

- Generar hidato aleatorio con parámetros
- Proponer hidato
- Jugar hidato
- Hacer movimiento
- Retroceder movimiento
- Consultar mejores puntuaciones hidato
- Ver resolución hidato
- Pedir pista
- Registrar usuario
- Login Usuario
- ~~Guardar hidato:~~
- ~~Consultar puntuaciones por dificultad~~
- ~~Guardar partida~~
- ~~Reanudar partida~~
- ~~Elegir hidatos guardados~~
- ~~Generar hidato aleatorio por nivel~~

Organización repositorio Github:

Enlace del repositorio: https://github.com/Alfredu/PROP
Carpetas:

- .idea : archivos xml del proyecto en IntelliJ
- docs: documentación del código en Javadoc y DiagramaDisseny.html
- src: contiene todo el código, tanto las clases del programa (carpeta main) como los tests unitarios de algunas clases críticas(carpeta test)

Nuestro algoritmo de resolución de Hidatos es un algoritmo de backtracking. Partiendo de la casilla 1, en cada iteración buscamos si el siguiente número a colocar se encuentra

adyacente a la casilla donde estamos. Si es así, se prosigue por ese camino. Si no se encuentra, se prueba en cada una de las celdas vacías adyacentes a la nuestra.

Este algoritmo funciona suficientemente bien para casos reducidos, pero queda claro que será necesario aplicar algún tipo de heurística para mejorar el rendimiento en los Hidatos más grandes o complejos.

Para generar Hidatos aleatorios nuestra implementación actual es la siguiente:

- Se pide el número de casillas total del Hidato, el número de casillas *agujero* (aquellas casillas visibles pero en las que no se puede colocar ningún numero) y el número de casillas con valor fijo.
- Se colocan en el tablero las casillas agujero aleatoriamente.
- Se coloca en el tablero la casilla 1 aleatoriamente. Se comprueba si el Hidato resultante tiene solución. En caso afirmativo seguimos. Si no, volvemos a empezar el proceso
- Asignamos el numero de casillas requerido menos una (porque siempre se incluye el numero 1) de la solucion como casillas de valor fijo. Vaciamos el resto.

Igual que anteriormente, esta implementación no es óptima y admite muchas mejoras para evitar situaciones donde no hace falta comprobar todo el Hidato para saber que ese tablero no es solucionable. Particularmente el algoritmo empieza a tardar un tiempo inaceptable a partir de un tamaño de tablero aproximado de 50 casillas.