EDA

ALGORISMES

Generació, validació i resolució:

Per aquestes tres funcionalitats principals (totes a la classe Hidato: solve(), valid(), generateHidato()...) es fa servir un algorisme de backtracking amb certes optimitzacions o condicions:

- A l'hora de posar el següent valor que pertoca, es comprova que no es repeteixen o es salten números dels donats al hidato inicialment.
- Comprovació de graf connex: bàsicament aquest és un problema de buscar un camí hamiltonià, el qual no és possible si el graf és disjunt. Per tal de comprovar això es fa una funció bfs que intenta passar per tots els vèrtex

Per emmagatzemar l'hidato i fer els càlculs necessaris bàsicament tenim una matriu de Cella (representa cada casella), un vector given (conté valors ja assignats) i start (coordenades de la posició inicial).

Check:

Per a comprovar si una solució és correcta és obvi que no ens cal backtracking, simplement es comença per l'1 i buscant per caselles adjacents busques el següent número fins a a arribar al final.

Jugar i proposar:

Aquí tenim la interacció usuari-hidato, la cual es guarda a la classe Partida a una pila de Log. En aquesta pila hi ha les coordenades on l'usuari ha posat un número per poder tirar endavant i endarrere a voluntat al llarg de la partida. Cal tenir en compte que el joc fa que haguem d'actualitzar atributs d'hidato com given o altres per tal de poder resoldre l'hidato o comprovar si tal i com està jugant l'usuari es pot arribar mai a una solució.

POLIMORFISME

Taulell i Cella:

En aquestes dues classes hem fet servir el patró "Decorator". Ambdues classes són abstractes i a través del adapter les podem inicialitzar com a diferents tipus de taulell o cella. D'aquesta manera, mètodes com getAdjacencies(), generateTaulell(), etc no cal repetir-los tres vegades sinó que, en funció de quina subclasse sigui, es cridarà una funció o una altra.