Algorisme

Per implementar l'algorisme ens hem basat en el paper *Efficient solutions for Mastermind using genetic algorithms*, desenvolupat pel *Research Centre for Operation Research and Business Statistics* de Leuven, Bèlgica. L'algorisme expressat en pseudo-codi és el següent:

```
Set i=1;
Play fixed initial guess g( 1 );
Get response X( 1 ) and Y( 1 );
while ( X( i ) != P ) do
        i = i + 1;
        set E( i ) = { } and h = 1;
        Initialize population;
        while ( h <= maxGen and |E( i )| <= maxSize ) do
                Generate new population using crossover, mutation, inversion and permutation;
                Calculate fitness;
                Add eligible combinations to E( i ) if not already in E( i );
                h = h + 1;
        end while
        Play guess g( i ) ∈ E( i );
        Get response X( i ) and Y( i );
end while
```

On:

g(i):	Codi a provar en el torn i.
X(i) , Y(i):	Nombre de fitxes blanques i negres, respectivament, obtingudes al comparar g(i) amb el codi Objectiu.
E(i):	Conjunt de codis candidats a ser el codi test en la iteració i.
h:	Nombre de generacions de codi realitzades.
maxGen:	Limit de generacions per iteració, per raons de temps de computació
maxSize:	Limit de tamany d'E(i) per iteració, per raons de temps de computació
P:	Output desitjat en el cas d'encertar el codi objectiu (N=4).

El funcionament de l'algorisme és el següent:

El primer torn l'algorisme sempre fa la mateixa suposició (0123 pel nivell de dificultat fàcil, 0012 pel nivell mitjà i 00011 pel nivell difícul), de la qual rep un feedback X(1) i Y(1) generat pel jugador, que representa el nombre de pins blancs i negres respectivament.

A partir d'aquí el codebreaker entra en el bucle principal, que s'executa sempre i quan X(i) no sigui igual a la longitud del codi, és a dir, s'executa fins que troba el codi solució que concorda amb el model. Un cop dins del bucle es fan les inicialitzacions pertinents: s'augmenta el comptador de turns, es crea un set buit de codis elegibles, es genera aleatòriament una població de 150 codis i s'inicialitza el nombre de generacions permés en aquest torn.

El segon bucle (bucle interior) és el que realitza les modificacions genètiques als codis, calcula el valor de fitness de cada codi generat, per mitjà d'una comparació entre aquest i els codis jugats durant els torns anteriors, i finalment escull com elegibles aquells codis que tenen un fitness igual a zero (i no es troben ja en el set d'elegibles). Finalment escull un dels codis elegibles de forma aleatòria i demana al jugador el feedback un altre cop.

Hi ha una petita diferència entre el nostre algorisme i l'algorisme que plantejaven en el paper. En ell s'explica que un cop seleccionat el conjunt de candidats E(i) i de cara a seleccionar el codi test, s'agafa un subconjunt d'E(i) i es busca dins del subconjunt aquell codi que sigui més similar a la resta, de cara a agilitzar les següents iteracions. En el nostre cas hem considerat que l'optimització no és necessària ja que pels objectius del projecte el codi és prou eficient, de manera que un cop tenim el conjunt E(i) n'agafem un element qualsevol, considerant únicament que aquest no hagi estat seleccionat anteriorment com a codi test en una altra iteració.

Aquí teniu un enllaç al paper, per qualsevol dubte:
https://lirias.kuleuven.be/bitstream/123456789/184247/2/Mastermind.pdf

Estructures de dades

Pel que respecta a les estructures de dades utilitzades, aquestes són les estructures de dades genèriques utilitzades:

Set: Hem optat per utilitzar set a l'hora d'emmagatzemar conjunts de població, degut a la seva simplicitat i donat que no requeríem de forma notòria accedir a elements específics de la població. A més ens ha estat molt útil a l'hora de comprovar que no afegim elements ja inclosos al set, ja que la pròpia funció *exampleSet.add(exampleElement)* retorna false si *exampleElement* ja és a *exampleSet*.

ArrayList: Utilitzat de forma auxiliar només en casos concrets d'ordenació o necessitat d'accés puntual.

Integer: Exceptuant els casos de retorn o variables iteradores en els quals hem usat l'estructura **int**, en la resta de casos hem usat l'objecte **Integer** ja que compta amb moltes més funcionalitats.

Pair: Hem implementat l'estructura Pair ja que ens és molt útil tant a l'hora d'emmagatzemar el feedback (BN) dels codis testejats, com a l'hora de guardar conjuntament el codi amb el seu feedback respectiu: *Pair<String codi, Pair<Integer Negres, Integer Blanques> >*.

Seguidament fem especial émfasi en aquelles estructures de dades que s'usen a la classe codeGenerator, més concretament a la funció m_generateCodeTest, ja que són les més importants de cara a l'execució correcte i eficient de l'algorisme genètic.

guesses: Set que conté els torns anteriors, en format de Pair amb el codi com a primer element i el seu feedback respecte al codi objectiu com a segon. A l'inici de cada crida a la funció m_generateCodeObjective s'actualitza guesses amb l'últim torn, introduït com a paràmetre.

populationSet: Set d'inicialització de la població que s'usa a cada torn, a partir del qual es generen les modificacions genètiques. Un cop inicialitzada, la població està composta per 150 codis generats aleatòriament.

newSons: Es tracta d'un ArrayList de codes que s'usa com a estructura auxiliar, donat que realitzem les modificacions als codis de *populationSet* de dos en dos. El funcionament és el següent: els codis de *populationSet* són processats sota una probabilitat de .5 per cadascuna de les dues funcions crossover, i el resultat és emmagatzemat a newSons. Sobre *newSons* es criden les funcions mutate, permutate i inverse, i finalment s'afegeixen a *sons*. En cas que els codis de *newSons* ja fossin anteriorment a *sons*, se substitueixen per codis aleatoris amb l'objectiu de donar diversitat a la població.

sons: Conté la població de codis de *populationSet* un cop han estat degudament modificats per mitjà de crossover, mutació, permutació i inversió.

sonwfit: Set auxiliar que conté Pairs amb l'estructura següent: ***Pair<Code chosenCode,Integer fitness***. Un cop conté tots els codis de *sons* amb el seu respectiu fitness value, s'ordena en ordre creixent segons el fitness value i se seleccionen tots aquells codis que tenen un fitness value igual a 0. Un cop s'arriba al primer element amb un valor de fitness diferent, es para la cerca.

candidates: Un cop s'ha assignat als codis de *sons* el seu respectiu fitness value, a *chosenSons* s'afegeixen aquells codis que tinguin un fitness value igual a 0.

chosenSons: Finalment a *chosenSons* s'afegeixen els codis de *candidates* de la iteració pertinent que no siguin ja a *chosenSons*.

new_population: És una població nova de 150 elements formada pels codis de *candidates* i per codis aleatoris per donar diversitat.