Pràctica CAP Q1 curs 2020-21
(*Backtracking*)

- A realitzar en grups de **2 persones**.
- A entregar com a molt tard el diumenge **10 de gener de 2021**.

Descripció resumida:
tl;dr - Aquesta pràctica va de continuacions.
La pràctica de CAP d'enguany serà una investigació guiada del concepte general d'*estructura de control*, aprofitant les capacitats d'introspecció i intercessió que ens dóna Smalltalk. Estudiarem les conseqüències de poder *guardar la pila d'execució* (a la que tenim accés gràcies a la pseudo-variable **thisContext**) per fer-la servir i/o manipular-la. El fet de poder guardar i restaurar la pila d'execució d'un programa ens permet implementar *qualsevol* estructura de control i implementar la versió més flexible i general de les construccions que manipulen el flux de control d'un programa: les *continuacions*.

Material a entregar:
tl;dr - Amb l'entrega del codi que resol el problema que us poso a la pràctica NO n'hi ha prou. Cal entregar un informe i els tests que hagueu fet.
Haureu d'implementar el que us demano i entregar-me finalment un ***informe*** on m'explicareu què heu aprés, i ***com*** ho heu arribat a aprendre (és a dir, m'interessa especialment el codi lligat a les proves que heu fet per saber si la vostra pràctica és correcte). Les vostres respostes seran la demostració de que heu entés el que espero que entengueu. El format de l'informe és lliure, i el ***codi*** que m'heu d'entregar me'l podeu entregar de diverses maneres: via un fitxer .st obtingut d'un File Out o via un paquet .mcz. Qualsevol d'aquestes maneres és vàlida. Utilitzareu **Pharo 6.1** per fer la pràctica.

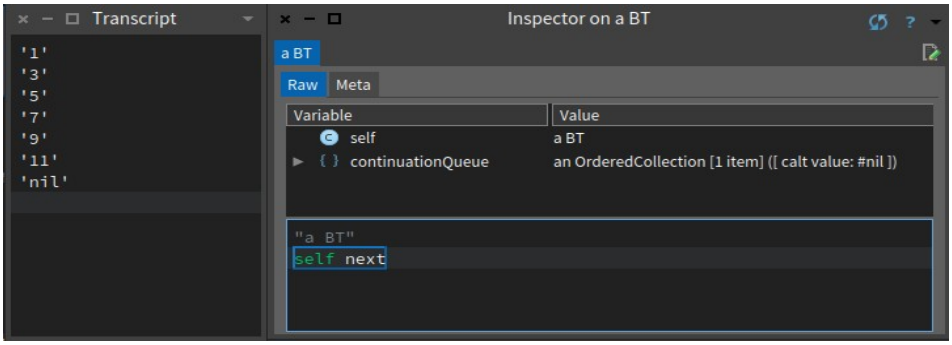
Enunciat:

Cal completar una classe **BT** (que vol dir *BackTracking*) tal que les seves instàncies es puguin fer servir per anar proporcionant valors d'una col·lecció a mida que siguin requerits. Tindrem dos mètodes, **#try:**, que requerirà una col·lecció com a paràmetre, i **#next**, que formaran el nucli de l'utilitat **BT**.

Per exemple, si executem al *workspace*:

```
| bt  x |
bt := BT new.
bt inspect.
x := bt try: { 1 . 3 . 5 . 7 . 9 . 11 }.
x asString traceCr.
```

Podem anar re-assignant un valor de la col·lecció **{ 1 . 3 . 5 . 7 . 9 . 11 }** a **x** i escrivint-lo al **Transcript** utilitzant l'inspector i enviant el missatge **#next** a l'objecte repetides vegades (tantes com elements té la col·lecció):



En acabar-se la col·lecció i enviar el missatge **#next** obtindrem **nil**.

Fixeu-vos que l'única cosa que hem fet per obtenir el resultat és enviar **#next** a l'objecte inspeccionat. No hem fet res més. Fixeu-vos també que **x asString traceCr.** s'ha executat sis vegades. Aquí és on haurieu d'entendre que calen continuacions per implementar **#try:**.

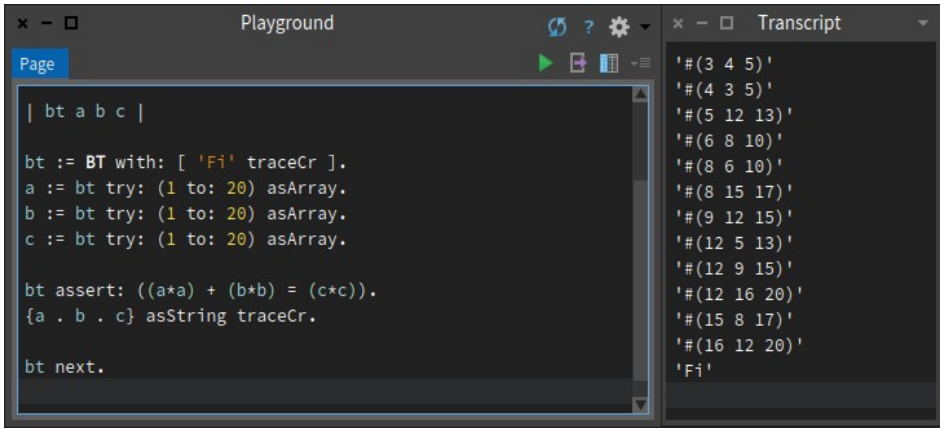
Un altre exemple. Suposem que executem aquest codi en un *playground* i demanem *print-it*:

```
| bt x y |
bt := BT new.
x := bt try: #(1 2 3 4).
y := bt try: #(5 4 3 8).
(x * 2) = y ifFalse: [ bt next ].
```

{ x . y } ⇨ **#(2 4)**

Enteneu el resultat? S'han triat dos valors d'**x** i de **y** tal que es satisfà la propietat **y = 2*x**.

Un altre exemple: Volem trobar tots els grups de tres números (a,b,c) menors que 20 tal que es satisfaci el teorema de Pitágoras ($a^2 + b^2 = c^2$), i que quan acabi escrigui '**Fi**' al **Transcript**



En acabar hauria de sortir un error (que no veieu al gràfic). Això té a veure amb com implementem **#try:**. És normal, ja que si demanem més dades quan ja les hem vist totes, el programa hauria de queixar-se.

Part 1: Us passo la implementació de la classe **BT**, excepte el mètode que cal implementar. En el codi font podeu veure com funciona la classe, i el nom de l'únic atribut que té ja és prou informatiu. En aquest enunciat podeu veure alguns exemples petits de l'ús de **BT**. La primera part d'aquesta pràctica és:

Implementeu BT >> try: aCollection

Part 2: Resoleu el problema de les N reines utilitzant la classe BT. No serveixen altres solucions que no exploten el mecanisme de *Backtracking* que BT ens proporciona. Per exemple, per trobar una solució per un tauler 8 x 8 hauriem de poder fer:

```
| res nq |
nq := NQueens with: 8.
res := nq solve.
res inspect.
```

Si volem totes les possibles solucions, caldria fer servir **#solveAll**. Els mètodes **#solve** i **#solveAll** són obligatoris: **#solve** proporciona una solució, **#solveAll** proporciona totes les solucions possibles (en el cas d'N=8 n'hi ha 92, vegeu http://www.ic-net.or.jp/home/takaken/e/queen/). Per implementar **#solve** en teniu prou amb els mètodes **#try:** i **#assert:** de **BT**. Per implementar **#solveAll** heu d'anar amb compte, *us caldran continuacions per controlar l'acabament*.

La solució tradicional d'aquest problema la podeu trobar a les planes 49-53 dels apunts d'EDA (https://www.cs.upc.edu/eda/data/uploads/eda-codis.pdf).