Revisting RandomSearch

Franck van Breugel

Java PathFinder (JPF) includes several search strategies. The best known are depth first search and breadth first search. It also includes a random search. This random search is implemented in the class RandomSearch, which is part of the package gov.nasa.jpf.search. According to its documentation,

> "this is a straight execution pseudo-search - it doesn't search at all (i.e. it doesn't backtrack), but just behaves like a 'normal' VM, going forward() until there is no next state then it restarts the search until it hits a certain number of paths executed"

However, even when run on a trivial Java app, this random search does not terminate. Consider, for example, the following app.

```
import java.util.Random;

public class Example {
   public static void main(String[] args) {
      Random random = new Random();
      System.out.println("0");
      if (random.nextBoolean()) {
         System.out.println("2");
      } else {
         System.out.println("1");
         if (random.nextBoolean()) {
            System.out.println("4");
         } else {
            System.out.println("3");
            if (random.nextBoolean()) {
               System.out.println("6");
            } else {
               System.out.println("5");
            }
         }
      }
   }
}
```

Run on the above app, JPF produces the following incomplete information.

```
JavaPathfinder core system v8.0 (rev 32+) - (C) 2005-2014 United States Government. All rights reserved.

============================================================ system under test
Example.main()

============================================================ search started: 2/15/18 1:17 PM
0
1
3
5
```

The aim of this project is to repair the implementation of this search strategy. Based on the material on search strategies (course slides and notes), the class RandomSearch is revisited. Furthermore, the framework to test search strategies, as developed during the lectures, is adapted so that it can be used to test the RandomSearch class.

Milestones

1. Revisit the RandomSearch class, fixing all its bugs.
2. Test the RandomSearch class.
3. Add parameters that allow the user to configure the random search.

The third milestone is optional.

Deliverables

1. RandomSearch class (fully documented).
2. Adaptation of the testing framework (fully documented).
3. Report that describes how the search can be used and configured.