Testing with self-learning and self-exploring testing tools

By Rik Marselis and Sander Mol, March 2017

Introduction

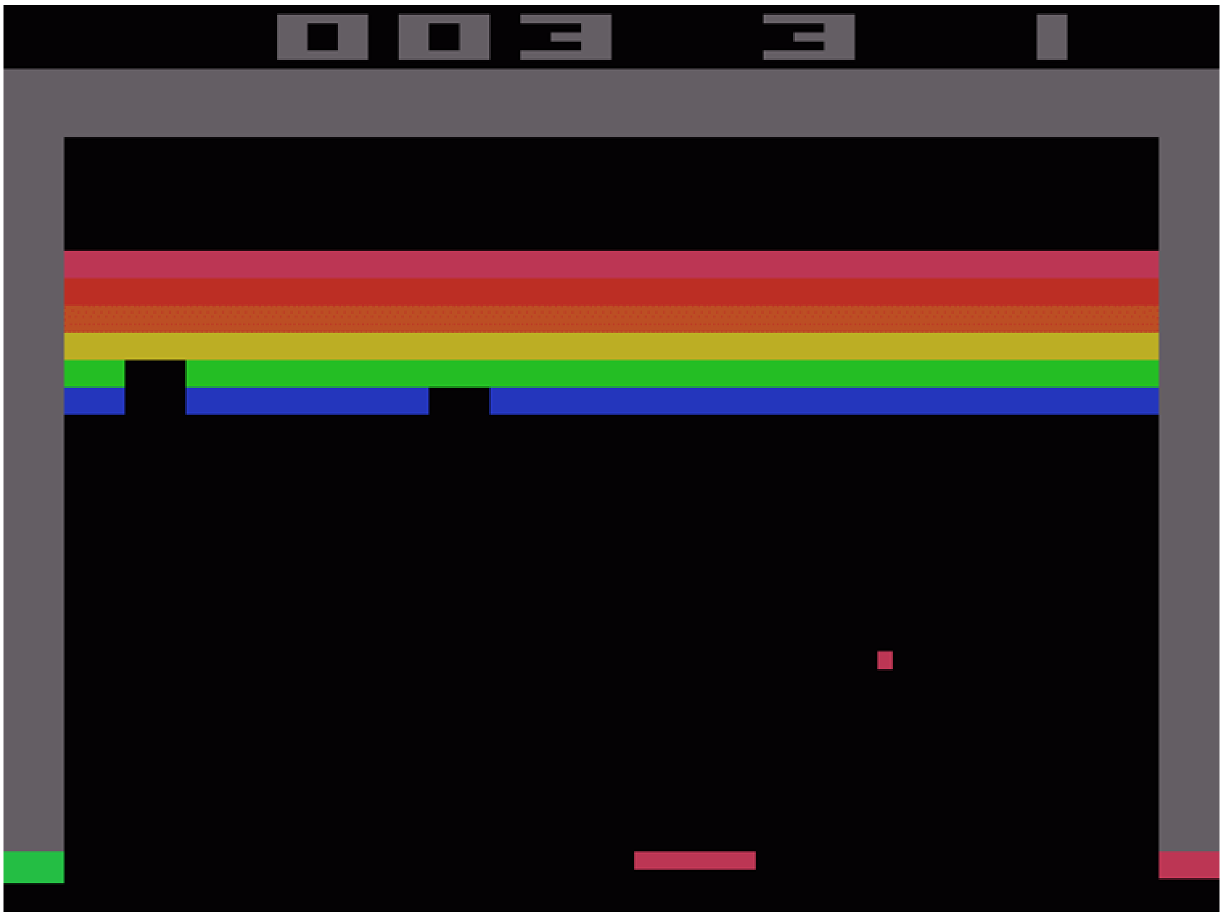
In what way will machine intelligence assist us as testers in the future? While we were discussing this matter for a while, we thought of the concept of self-learning and self-exploring testing tools. Below we sketch our ideas. And at the end of this article we call for you to explore this matter further with us!

Testing Tools of today are mainly to take over the (often boring) job of test execution

Test Automation usually automates test cases that have been previously compiled manually by the testers. These test cases have predefined expected outcomes. The tool verifies that the actual outcome matches the expected outcome. The test cases selected, aim to achieve a certain level of coverage of all possibilities. Changing the test object also means adjusting the test cases in the tool.The world of testing is now about to apply new, other types of automated testing tools. Tools that do not need to get instructions in advance, but independently explore an information system.

An introduction to the self-learning and self-exploring tools of today

The possibilities to have machines to learn autonomously have greatly increased. The world watched in astonishment how the self-learning AlphaGo defeated the Korean world champion of the game “Go” convincingly, 10 years earlier than expected. And now a computer has already won a lot of money from top players at the game of poker. The makers of AlphaGo, Google’s “DeepMind” also created another program that excels in self-learning and self-exploration. This program manages to investigate classic Atari games without any explanation, playing and learning it until it is played to perfection. As an example we choose the game breakout of which you see a screenshot below.



The DeepMind program starts the game and initially has no idea what is happening. The concept of a moving ball which needs to be bounced back is unknown. The same goes for the idea that the pink pixels at the bottom of the screen together form an object that can be controlled, and that the points are scored by playing off the colored blocks. Everything is new, so everything needs to be explored and learned.

This exploring and learning is done by playing the game time and time again. After 10 minutes of playing, the program will already have some skill in controlling the paddle. After 300 games, the program can play just as good as a human player could. And after 500 games, the program has found the best stragegy to win the game by making a hole at the side and let the ball bounce over the top row of blocks.With this tool other games can be learned as well. On Youtube you can find examples of Pacman and Super Mario. It’s played without explanation. With a single tool.

Self-learning and self-exploring testing tools take the over the thinking

Suppose we would not set this DeepMind tool to learn an Atari game, but an object that we want to test, such as a website. What could such a tool do? What could we achieve using the tool? Let us focus on what is achievable in the short term: input checks, layout verification, error handling, and here and there some tests of the consistency of values in two fields. The so-called syntactic and semantic tests.

The tool would start clicking around without any prior knowledge, do some typing, drag the cursor and so on. It would detect input fields and enter all sorts of values. It would make random combinations of inputs and actions and then experience what the results are. It would identify trends and patterns and thus determine what is standard behavior. And once the tool knows what the standard behavior is, it can also recognize deviant behavior. After doing all its exploration, the tool can generate a report with a list of these anomalies.

That is where we as human testers are still needed: to interpret the anomalies and to determine the difference between desired deviations and actual defects. Of course, the self-learning program will take this human input into consideration the next time, thus learning about desired behavior of the system. The next round of testing will result in a much shorter list of anomalies to interpret.

As with the current day test automation, the strength of this concept is in repeating the test. At the next release of the test object, the tool will recognize what the changes are and again make no distinction between right and wrong. The human tester thus can assess both the desired changes as well as any regression defects.

Besides, who says that this must be done by a professional tester? We can also ask other stakeholders such as customers, to evaluate the reports with the support and advice of the artificial intelligence.

Can we still help the tool with anything?

Above we limited ourselves to syntactic and semantic testing, but the move to automatic testing of processing logic and process flows is quite conceivable. Certainly if we help the tool a little by giving different 'inputs'. Ultimately, the tool can invent (almost) everything by itself after thousands of hours of exploring and learning. The main reason to supply some inputs is increasing the efficiency.

Some examples:

- We could supply test data, such as valid postal codes and correct existing addresses, the right format of telephone numbers and examples of existing usernames and passwords.
- We could supply designs and requirement specifications, preferably in a structured format. The tool will then try to identify which input and output values are in this test base, and try to validate the test object against it.
- We would, after a first round of tests, indicate which of the detected features we find most important, so that the tool puts its focus there in the next round of testing.

In addition there is something that we, as professional testers would like to add: test design techniques. This allows us to aim for a specific coverage. These techniques are often not applied in practice, so it would help a lot if a tool would facilitate this. Moreover, because the tool can basically do an infinite number of tests, artificially intelligent tools themselves can also come up with new techniques, perhaps even better than we can think of as humans. Just think of the trick described above where the tool itself learned to play the most effective game of Breakout. The limitation currently is mainly the time it takes for the tool to try out the many possibilities.

Can the tool help itself with anything?

In addition to the human 'inputs' there is another very interesting "input": the earlier tests of the tool itself. In this way, the tool can test much more efficiently during the next round of testing. Compare it to the 500th game of Breakout. The experience of the tool of course does not need to focus on just one test object. For example; if we have tested a website and we as human beings have indicated the type of variations we do not find desirable, at the second website the tool will reuse this experience. Think of quality characteristics like usability and accessibility. And if after some time we have tested (or actually explored) hundreds of websites and have had humans judging those (learning), then the tool recognizes general trends in expected behavior which the tool will take into account at the next test. Knowledge and experience about hundreds of test objects, fully available with the touch of a button. Will this be the end of the need for a professional functional tester's knowledge, skill and years of experience?!

How feasible is this concept?

Let's take a deep breath first. At present, self-learning and self-exploring testing tools still are fictitious. But all the necessary knowledge and techniques already exist at this moment. An important question of course is whether the investment in this type of testing tools outweighs the results. In particular the adoption of the creative thinking work of the human tester still sounds very futuristic, but the developments go very fast.

The first step will be that the testing tools do some of the exploratory work and thus provide a basis on which the human tester can elaborate. But for the longer future, it is not inconceivable that this kind of self-learning and self-exploring tools will be able to determine the quality of an information system in such way that those involved will gain sufficient confidence to take the information system to use.

Do you want to explore the future with us?

We'd like to continue developing this concept, along with fellow testers who like to join us in thinking about this. The possibilities seem endless. We already think big, and still we'll probably be surprised in the future. But there are plenty of obstacles to overcome. There's plenty to discuss about this look into the future! Are you interested, let us know!!

Furthermore, you may like to know that the code of the DeepMind program, to play Atari games, is publicly available on github. The programming language is C ++, so if you are familiar with this and want to contribute to this experiment, we want to hear from you!