

# **TOURING TURING**

*An Introduction to visual design and programming*

Interaction Design Institute Ivrea, Fall 2004  
STUDIO 1 with Walter Aprile & Britta Boland

## Touring Turing

This course consists of an introduction to graphic design and programming with a particular focus on building interfaces. The course has a skill section, where particular techniques and technologies are introduced, and a design brief that is developed throughout the course and delivered at the end.

**The object of the brief is to design and code a Graphic User Interface (GUI) to the Turing machine.**



## Turing who?

Alan Mathison Turing (1912–1954) was a British mathematician, logician, and cryptographer. He is one of the fathers of computer science. From his childhood it was clear that he was gifted for science; he studied at Cambridge under G. H. Hardy and at Princeton under Alonzo Church. In Cambridge he also met Wittgenstein, with whom he disagreed over the value of formalism. Turing's main contribution to modern science and thinking is an influential formalization of the concept of algorithm and computation: the Turing machine. He formulated the widely accepted version of the Church-Turing thesis, namely that any practical computing model has the capabilities of a Turing machine. As a matter of fact, your PC is equivalent to a Turing machine, despite the fact that a Turing machine is very simple and your PC is very complex.

This brief is drawn from chapter 3.3 of *School Again*, a book currently being written by Walter Aprile, Britta Boland & Stefano Mirti with Dario Buzzini and external contributors. Chapter 3.3. was done in collaboration with Daniele Mancini.

*“School Again* is a textbook for students of Design, centering on the intentionality of projects. We realize that, under the current conditions of extremely rapid technology change and innovation, we cannot center on specific technologies or skills. What connects the disparate disciplines and fields of study that we come from and that we have worked in, is that there is a stated goal that directs the various activities. We believe that, in learning by doing, the point is not the specific techniques: the point is the intentional attitude and process. This attitude and this process constitute the object of our book.”

*(from the back flap)*

During World War II he worked on breaking German ciphers, particularly the Enigma machine; he remained throughout the war the chief cryptanalyst for the Naval Enigma effort – this was kept secret until the 1970s.

After the war, he designed one of the earliest electronic programmable digital computers at the National Physical Laboratory and, shortly thereafter, actually built another early machine at the University of Manchester. He also made significant and characteristically provocative contributions to the discussion “Can machines think?” – his “Turing test” is an operational definition of intelligence, still widely discussed in the fields of philosophy and artificial intelligence. He also did significant work on pattern formation and mathematical biology.

If Turing’s professional life held many impressive achievements (some of which he had to keep secret), his personal life was not very happy. At school his single-minded passion for the sciences clashed with the educational model. The great public disaster of his life, though, happened in his last years: Turing was homosexual, and at that time being homosexual was a crime in Great Britain. Tried and indicted in 1952, he was forced to choose between incarceration and hormone treatment. He chose the latter, which was not without physical and psychological effects.

Alan Turing died in 1954 of cyanide poisoning, apparently from a cyanide laced apple. While the police investigation ruled this to be suicide, speculation about this unusual and grisly death still abound.

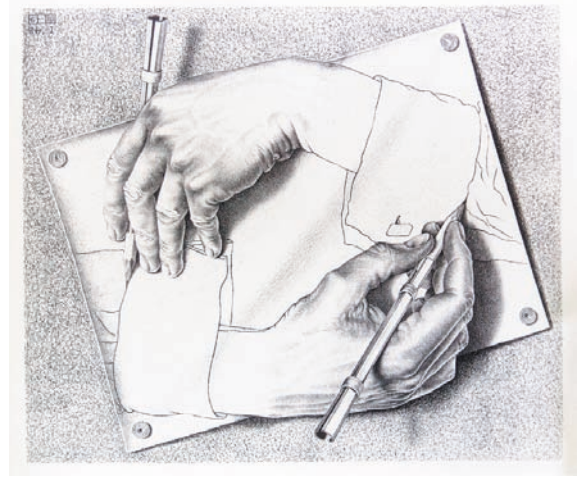
*(modified from Wikipedia)*

## Self-Reflection

The Turing machine is really a way for computing to reflect upon itself, on its limits and on its possibilities. It fits in a tradition of self-reflection that is represented in the arts all through Renaissance up to the works of M.C. Escher.

Escher’s works have immediate appeal, even outside the province of the students of science. Their curious construction and their cleverness can be enjoyed by everybody.

Still, they can also be seen as illustrations of profound mathematical and geometrical concepts.



(M.C. Escher, *Drawing Hands*, 1948)

But they don’t need to be completely understood to be enjoyed.

## What is a Turing Machine?

Informally, a Turing machine is composed of an infinite tape over which there is a moveable head, like a tape head in a tape recorder.

The tape is composed of cells that contain a symbol taken from an alphabet.

The head can move in both directions, but only one cell at a time and it can read from the tape and write to it. The head can read and write only the cell that it is on. Furthermore, the head has an internal state, taken from a set of acceptable states.

Usually states are indicated by numbers, and the alphabet for the tape is made of letters.

Additionally, there is a table of rules that the head follows. This would be a typical rule:

If the head is in state 2, and it is reading the letter A from the tape, then change the state to 1, write a B and move right.

In the interest of compactness, rules are written using a shorter form like this one:

(state now, read symbol, new state, write symbol, direction)

The previous rule would then become

(2,A,1,B,>)

All the machine does is this: read the tape, check if there is a rule that applies, apply the rule (that’s to say: change state, write the symbol and move). If there is no applicable rule, the machine stops. A

complete description of the machine also includes an initial state for the head, drawn from the set of acceptable states, and a initial condition for the tape, defined as an infinite sequence of symbols taken from the alphabet.

## Please, make it stop

Nowhere in the definition it is stated that the machine has to stop eventually. In fact, there is no guarantee that it will, and Turing himself proved that it is impossible to predict whether the machine will stop. In other words, to find out if the machine will stop, you need to run it until it stops.

The inability to predict stopping has enormous consequences: remember that Turing's work shows that to say "Turing machine" is the same as saying "program" or "piece of software". In other words, a programmer cannot reliably look at a program and know if it will ever stop running or not – which means that a program can only be debugged by running it.

It is true that a real computer has much more hardware than what makes up a Turing machine: a real-world computer often has a display, a keyboard, some form of removable storage, a network connection and countless other things that you can mail order. But all these things, while they are of great importance to the user, do not change what the computer can or cannot do – it does not matter how much RAM you add, or how fast your processor is: you still will not be able to predict if a program will stop or not.

When we say that computers are Turing machines we mean that they share the same limits.

Another practical implication is this one: suppose that you have a program that takes JPEG images and turns them into GIFs or some other format. Your friend tells you "I have a better program: it is faster and it does exactly the same thing". But we know that, while you can run your friend's program on any number of test cases, there is no formal way to prove that his program is equivalent to yours – that's to say, you cannot look at two pieces of code and state that they do exactly the same thing.

## The Turing Tarpit

The Turing machine is beautifully minimal, and it can be used to prove many things. On the other hand, it is a very unpractical tool for everyday programming because it requires endless attention to detail and forces you to deal with one symbol at a time. We have invented programming languages precisely in order to have a friendlier way to write useful.

Referring to the Turing machine's universality and extreme unpracticality, a computer scientist named Alan Perlis wrote "Beware of the Turing tarpit in which everything is possible but nothing of interest is easy". Still, it is stimulating to think about some simple programs that could be written for the Turing Machine.

## Secret Messages

Suppose that you have a tape that contains a secret message. If you want to keep nosy people from reading the message, you can use a simple cipher, for example Caesar's cipher that transforms every letter into the one that comes two steps afterwards in the Roman alphabet. Thus A turns into C, B into D and –to close the circle– Y into A and Z into B. The rules for a Turing machine that performs this operation are quite trivial;

(1,A,1,C,>)

(1,B,1,D,>)

...you can fill this in yourself...

(1,Y,1,A,>)

(1,Z,1,A,>)

(1,.,0,..,>)

The last rule says, when reading a "." that signals the end of a message, rewrite the "." intact, and enter state 0. As there are no rules for state 0, the machine stops.

This machine works, but it doesn't do anything spectacular: it just zips through the tape swapping letters. During the course we will look at other, more complex, examples to see how interesting results can emerge from simple sets of rules.

## Desiderata

The Turing machine is the simplest, cleanest, conceptual model of a computer. Its rules and operation can be explained in a half page, but there is endless variety in the behaviors it can manifest.

Despite its simplicity, it can do everything<sup>1</sup>. As a matter of fact, mathematicians currently define “effectively computable” as “computable by a Turing machine”. The Turing machine is like a computing benchmark: for example, JavaScript, C, cellular automata and Python are all equivalent to Turing machines – but HTML is not, nor is Excel. In practice means that you cannot program in HTML, and that by choosing C over Python you are just choosing between two equally expressive languages to say the “same” thing.

At the same time, the Turing machine is not widely known or understood (or played with!) outside of the computer science world. Contrast this with the broader currency enjoyed by ideas such as cellular automata, in the particular case of the game of Life.

## Back to the brief

**The object of this brief is to make a GUI to the Turing machine.** The GUI must visualize the operation of the Turing machine, allow the user to modify the machine (rules and table) and its tape, and entice the novice user into exploration. The characteristics that we are looking for are:

It must be intriguing and fascinating (since the TM is intriguing and fascinating, its interface and representation must be too)

It should be attractive to a novice

A novice should be able to grasp it

An expert should not be hindered by it

It must support explorative behavior, such as changing the machine or its tape on the fly

It should encourage exploration by supporting undo and the storage/loading of states

It should be generic enough to edit, run and display various specific Turing machines

It should address the problem of **time**

The last point bears explanation: a Turing machine is a process that takes place over time. The state of the machine changes, the head moves and the tape is modified. Is there a way to show how the process evolves, so that endless loops, machine stops and starts, and what happens generally is made visible and understandable?

## Final deliverables

**By Thursday 28th October you will be expected to deliver:**

A piece of Processing code with its attendant resources, debugged, commented and in working order, to be presented on the 28th

The web-ready version of the Processing code, posted on the course website

Documentation for the code, in the form of a printed piece, one to two A4 pages in size, to be handed out on the 28th

a presentation, in which you explain your design choices, the unresolved issues, the possibilities for expansion and the difficulties you encountered. Be prepared to answer audience questions.

## Mid Crit

A moderately formal intermediate critique, known confidentially as a “mid crit”, will be held at the beginning of the third week of the course.

## Documentation

The documentation on programming under Processing is a subset of:  
<http://zope.interaction-ivrea.it/tc>

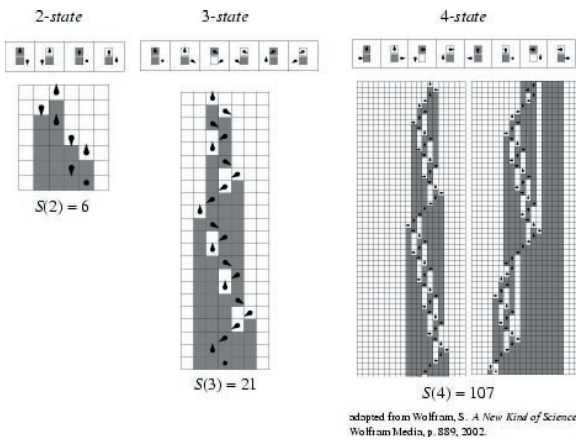
## Calendar

The course lasts from October 4th to October 28th. A detailed calendar is available on line. STUDIO 1 classes are held Monday and Thursday mornings from 9:30 - 13:00 in Comunità.

<sup>1</sup> For an appropriate definition of everything.

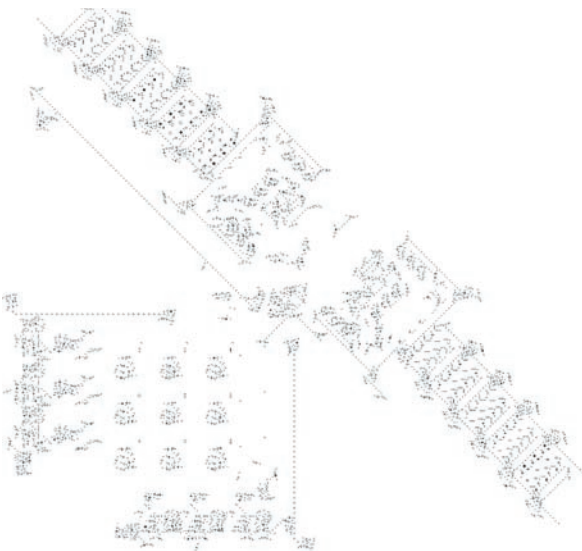
## State of the art

As a basis for discussion, we can see that available representation and interfaces to Turing machines are either horrible or difficult to understand. Witness the following example:



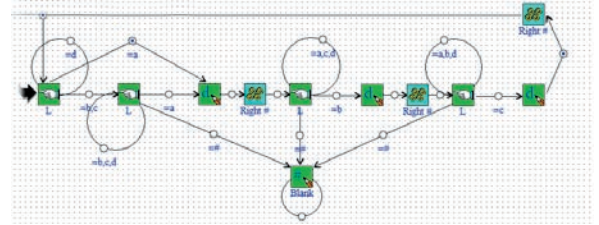
in this notation, used by Stephen Wolfram, it takes quite some intuition to understand what the little squiggles mean. But the representation of evolution in time is interesting.

A traditional Turing machine is more or less one-dimensional. By going to a two-dimensional representation, we can show time (just like a movie adds the third dimension of time to a sequence of two-dimensional images.)



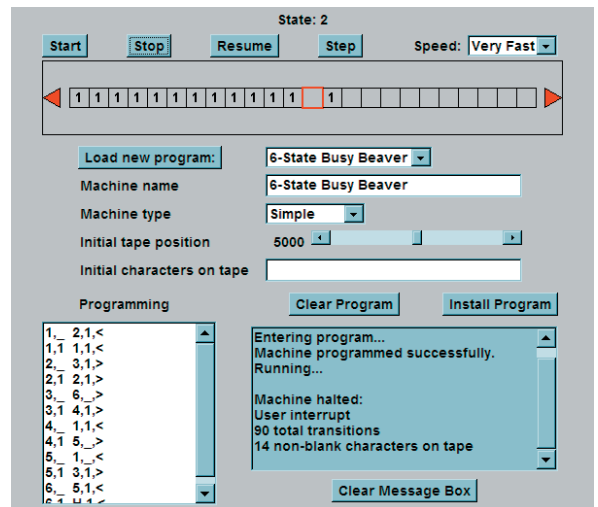
This is a simple Turing machine implemented as a configuration of the "Life" cellular automaton. A very scary concept. No interface here, other than pointing and clicking the cells whose state you want to change.

Not to get sidetracked – but there are even two-dimensional Turing machines, called *turmites*. They roam on a plane of cells. Of course, once you learn the trick of the higher dimension, it is difficult to stop. Enter n-dimensional Turing machines. But they are not more powerful than a normal one-dimensional Turing machine. How would you build an interface to those?



This is an alternative representation of a Turing machine, in the guise of an automaton. While it is more understandable than Wolfram's, we lose the idea of states.

This screenshot is from a Windows program called Visual Turing. Generally speaking, it is a very flexible program, although it looks ugly.



This simulator works, but it gets almost everything wrong in terms of interface. You cannot play with the tape, it is very easy to mistype a rule. Still, this applet would satisfy the brief.

Among the things that can be improved, we can say that there is no representation of time and, if the tape becomes very long, it is impossible to understand what is on it.

More profoundly, the metaphor of the "front panel" of a machine is very strong.

You can do better than this...

## **Bibliography & Links**

### **On Alan Turing:**

<http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Turing.html>

Andrew Hodges, Alan Turing: The Enigma

<http://www.turing.org.uk/turing/>

### **On Turing machines:**

(very compact)

<http://mathworld.wolfram.com/TuringMachine.html>

(extensive)

[http://en.wikipedia.org/wiki/Turing\\_machine](http://en.wikipedia.org/wiki/Turing_machine)

### **Simulators:**

Visual Turing, a TM simulator for Windows:

<http://www.cheransoft.com/vturing/>

Deus Ex Machine, another TM simulator for

Windows

<http://www.ics.uci.edu/~savoIU/dem/>

JFLAP, Java multiplatform Turing simulator

<http://www.cs.duke.edu/~rodger/tools/jflap/>

A Java TM simulator

<http://wap03.informatik.fh-wiesbaden.de/weber1/turing/index.html>

A solution to your brief, but as designed by CS student. Interesting documentation of the process:

<http://www.cs.utah.edu/~dhenders/cs4500/>