A Study of Empirical Modelling with Application to Historical Information Processing Methods, Using an Example of Punch Cards in *Cadence*.

0820850

January 31, 2012

Abstract

This paper will investigate how Empirical Modelling could be used as a way of teaching how information was processed in previous generations, using historical methods and devices. Computer programming has changed over time with the evolution of computers, and to this day, many of the old programming methods are still in use — directly and indirectly. However, the very first mechanical and electronic computers used a very different, hands-on approach to programming, involving much interaction with the hardware itself. Programs would be fed into the computer using punch cards — physical cards that contained encoded information in the form of holes. Later computers would use similar mechanisms such as punched tape, or physical manipulation of the machine's own switches and cables. These machines are no longer readily produced or available, and as such it is not easy to have a practical investigation or experimentation of the methods that were used to program the very first computers. The paper will explore the feasibility of using Empirical Modelling to provide representations of punch cards, to teach users the basis of historical programming methods and to allow them to experiment and learn from these models, without needing access to the hardware itself.

1 Introduction

The origin of modern-day computers may be traced back to 1837, when Charles Babbage described a design for what became known as the Analytical Engine. It was intended as a successor to his Difference Engine, of which an Empirical Modelling study was previously published [2006]. Babbage was never able to complete construction of his Analytical Machine [Weber, 2000]. The Analytical Machine was designed such that the input would be in the form of punch cards [Kopplin, 2002], and the proposed method of using punch cards as input is something that was utilised by early digital computers in the twentieth century. The punch cards were constructed such that a combination of holes punched in pre-defined positions of the card could be decoded by the computer and interpreted as information [Fisk, 2005]. It was possible for machines to mechanically produce punch cards, but this was not always the method put into practice, often being carried out by humans — some of the earliest programmers. Punch cards were programmed using keypunches, which over time evolved from being a manual hand-operated device, to having a typewriter-like keyboard used to semi-automate the process [Truedsell, 1965]. Programming with punch cards was often comprised of two phases — the initial patterns for the punch cards would be designed by programmers, who would pass these patterns to the keypunch operators for mass production. Punch cards defined a programming language or structure that is very similar to today's assembly language.

1.1 Hollerith Card

There have been various different implementations of punch cards since their inception. One of the earliest formats, and the format from which subsequent generations of cards are based, is the Hollerith card, named after its developer Herman Hollerith. The original Hollerith card was used for data collection

and processing for the 1890 US Census.

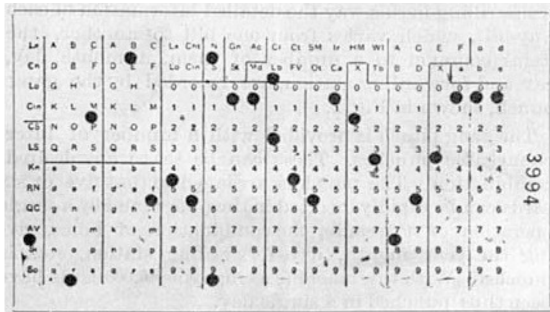

Figure 1: Original Hollerith card from 1890. *(Image: public domain.)*

Later generations of punch cards were adapted for generic processing and used a system of twelve rows — from top to bottom: rows 12, 11, 0, 1 through 9. For each of the columns of the punch card, punches in the zone portion of the card (rows 12, 11, and 0) combined with punches in the numeric portion of the card (rows 1 through 9) would represent either an alphanumeric or special character [Jones].

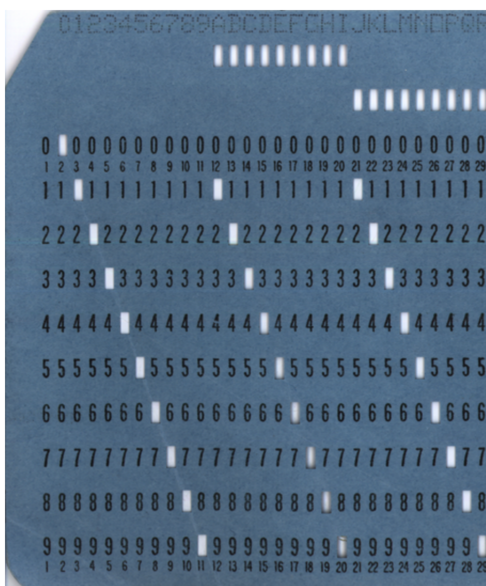

Figure 2: Left portion of an 80-column punch card, using the zone/numeric row system, and showing punch codes for characters 0-9 and A-R. Rows 12 and 11 are shown but not identified. The 80-column punch card left behind a legacy in that most character terminals display 80 characters in one line. *(Image: public domain.)*

Alpha characters from A to I are represented by punching row 12 and one numeric row, e.g. A is represented by punching rows 12 and 1.

Alpha characters from J to R are represented by punching row 11 and one numeric row, e.g. J is represented by punching rows 11 and 1.

Alpha characters from S to Z are represented by punching row 0 and one numeric row *from 2 to 9*, e.g. S is represented by punching rows 0 and 2.

Numerals from 0 to 9 are represented by punching the corresponding numeric row only, e.g. 1 is represented by punching row 1. Note that 0 is a special case, and is represented by punching row 0 only.

Special characters are represented by punching zero or one zone row and two numeric rows. Note that the forward slash character is a special case, and is represented by punching rows 0 and 1.

For the purposes of this study, the format of the original 1890 Hollerith card — with 3 zone rows, 9 numeric rows, and 24 columns — will be used. The README document within the model's files lists how each alphanumeric or special character can be represented on a Hollerith card.

1.2 Empirical Modelling

Empirical Modelling (EM) concentrates on the key concepts of observation, dependency, and agency. Russ [1997] compared these three concepts to features of a spreadsheet — the observables being the cells in the spreadsheet, which will be observed by the user; the dependencies being the formulae contained within the spreadsheet; and the agents being devices that somehow initiate a change of state, for example the user who is interacting with the spreadsheet. These concepts come together to form the empirical aspect — modelling through experimentation.

In the context of this study, an observable may be the state of the punch card, such as whether a hole has been punched in a certain position or not. The dependencies decide how the various punches combine together to produce an output, i.e. the character encoded by each column of the card. In learning how punch cards operate, and how to create the correct combinations of punches to generate the desired output, the instructional approach may prove to

be slow and inefficient. Through experiential learning, the user makes interactions between their ideas and their experiences with the model. From this experimentation, they gain a better working knowledge of the technology.

1.3 Motivation

Although a number of punch cards and punch card readers still exist today — and indeed are still used by some voting machines — they are no longer in mass production and prove difficult to obtain. As is especially the case when the user wants to "practise" programming punch cards, usage of existing punch cards may be seen as a waste of a rather valuable resource. Educational technology is one of the principle areas of application for Empirical Modelling [Beynon, 1997], and so Empirical Modelling hereby presents itself as a tool for providing the opportunity to learn about this largely-historical technology.

1.4 Existing Tools

There exist a small number of (mainly Internet-based) punch card tools, however it appears that the focus on such tools is in ease-of-programming rather than user experimentation. For example, Kloth [2003] offers a web-based applet where users can type in the text they wish to encode on a punch card, and the resulting punch card will be presented to the user in image form. Few (if any) tools exist that allow the user to experiment with punch cards by manually selecting which holes should be punched to reach their desired encoding.

1.5 Related Work

There have been several previous Empirical Modelling studies relating to historical computing, including *Modelling Babbage's Difference Engine* [2006] and *An Investigation Into the Empirical Modelling of Physical Devices, in an Educational Context, Illustrated with the Enigma Machine Example* [2006].

The former stated that the Difference Engine is "commonly considered to be the world's first computer", and thus it is important for computer scientists (and members of similar fields) to recognise and understand the basis of how it works. The study found that whilst the model created does mimic the functions of the Difference Engine, its inner operations (what's "inside the box") are vastly different from the design of the Difference Engine itself. Although this does not pose a problem if the aim of the model is simply to demonstrate the Difference Engine in action, it is not ideal from an educational point of view where the user may want to understand the mechanics behind the engine.

The latter focussed somewhat on the constructionist approach to learning — the act of learning by making things — and how the reverse process of disassembling the objects can also contribute to constructionism. The study developed a prototype of the Enigma Machine which allowed the user to interact with the model from a number of different perspectives and to set environment variables themselves, lending itself to the constructionist and experiential approaches of learning.

2 Modelling

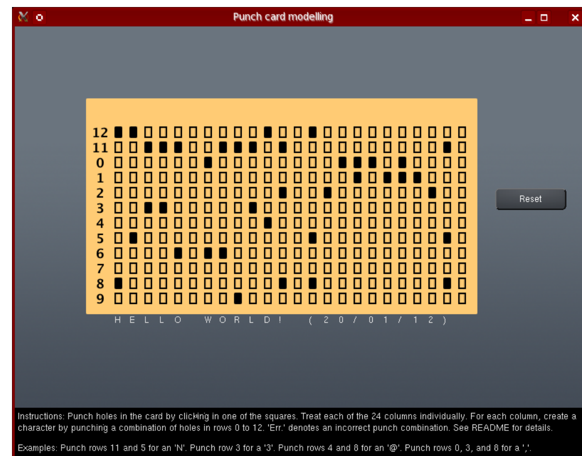

Figure 3: Overview of the model.

2.1 *Cadence*

This study uses *Cadence*, a prototype tool for Empirical Modelling. *DOSTE* (*Cadence*'s interpreter) contrasts with *EDEN* in that it aims to take a snapshot of the current environment state as a graph of interconnected nodes.

During implementation and testing, it became apparent that *Cadence* would crash and quit under certain conditions. Specifically, it appears that when browsing the node tree within the *Cadence* interface, attempting to expand a node which was subject to a high number of **if ... else ...** statements, the program

would quit with a segmentation fault or runtime error, usually relating to memory. This can probably be attributed to the fact that *Cadence* is still a prototype in development, and thus is not yet fully stable. The code for the model contains workarounds that try to minimise the number of **else** statements within one conditional, in an attempt to minimise these crashes whilst using the node tree browser.

2.2 Aims and Assumptions

The aim was to create a realistic punch card model that mirrors real-world behaviour as best as possible. It was therefore not in our interests to moderate the user's choice of holes to punch, for example by "disabling" incorrect combinations in some way. Rather, the user should have full control over what they choose to do within realistic bounds of real life punch cards. For instance, if the user wants to punch a hole in every possible slot on the card then they should be able to do this, regardless of the fact that the user will not get any meaningful encoding out of it. It should also be noted that in real life, users are not constrained to punching holes in pre-defined positions, and can in fact punch holes in any location and in any orientation. For simplicity and due to various limitations of *Cadence*, it will be assumed for the model that users can only choose to punch holes in 288 pre-defined positions.

To make the model as true to real life as possible, we need to consider that a single hole of a punch card cannot be "unpunched" — that is to say, once a hole has been punched, it will remain punched forever. This means if a mistake is made, the only way to rectify the error is to start again with a new card. Thus, although it may seem inconvenient or "harsh", the model should have no "undo" button, or facility to reset a single column individually.

It is assumed that the card is "read" from left-to-right, as in real life. It is also assumed that a blank column can be interpreted as a space, and all other valid punch combinations make up the EBCDIC character set.

2.3 Implementation

The main item of the model is the card, represented as a yellow box with 288 possible positions for holes to be placed (12 rows $\times$ 24 columns). Punched holes are shown as black boxes, and unpunched slots as "transparent" boxes with a black border. Each of the 288 slots are clickable, but only once — as mentioned earlier, this is a one-way process and "unpunching" a hole is not possible without resetting the entire card. Each of the 24 columns of the card are all treated independently, and the state of one does not affect the state of another in any way.

As stated previously, the rows are split into two sections — the zone portion (rows 12, 11, and 0), and the numeric portion (rows 1 through 9). Each of these twelve rows are given a unique weight such that, when adding together the weights of all the punched rows in a valid combination, a unique final weighting will be generated. It is this weighting that is held in a lookup table against all the possible characters, to determine which character the column in question represents. As way of an example, see the figure below:

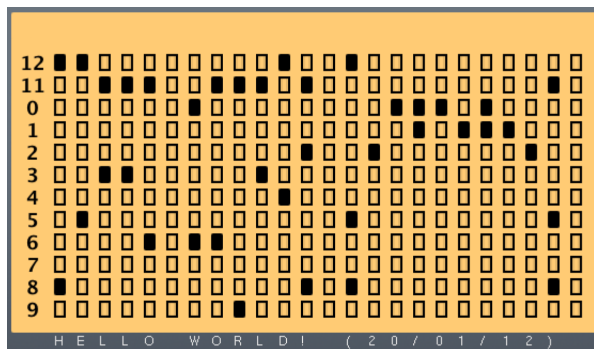

Figure 4: A punch card displaying punch combinations for the string "HELLO WORLD! (20/01/12)".

Looking at the encoding for the character "H", we can see that it is generated from a punch in rows 12 and 8. The weight assigned to row 12 is **13**, and the weight assigned to row 8 is **8**. The unique final weighting generated from adding the weights of these rows together is therefore $\mathbf{13}+\mathbf{8}=\mathbf{21}$, which corresponds to the character "H" in the lookup table.

An interesting special case is what occurs when there are punches in two rows of the numeric portion of the card, as is usually the case for special characters. Assuming that the weights assigned to rows 1 through 9 are **1** to **9**, we can see that there are multiple punch combinations that may add together to give the same final weighting. For example, a punch in row 1 and row 8 would give a total weighting of **9**, as would a punch in row 9 alone. It is clear that this poses a problem for the lookup table of values and characters, as different punch combinations no

longer generate unique lookup weightings. To combat this, the weight we assign to an individual row actually depends on whether there are punches in any of the other rows. We therefore say, for example, that the weight of row 8 is **8** *if and only if* none of the other numeric rows are punched. If (say) row 2 is also punched, then rows 2 and 8 gain a "combined" arbitrary weight of **280**. This new weight for the numeric portion is added to the weight for the zone portion, to produce the final, unique weighting used to look up the character required. The final lookup weightings are in the range of **0** to **998**, though not all weightings within this range are assigned to a character. This does not matter, however, as we maintain a one-to-one weighting-to-character relationship, rather than a many-to-one or one-to-many relationship.

It is also worth noting that incorrect punch combinations are assigned a weight of **999**. For example, punching row 3 and row 4 together is an invalid combination and as such will create a numeric total weight of **999**, which will be added to the weight of the zone portion of the card. Any final weighting of **999** or above generates an error, displayed as "Err." on the interface, meaning that the user has made an incorrect punch combination for a specified column.

2.4 Limitations and Extendability

One limitation of the model in its current form is the inability to model a sequence of punch cards to create a continuous stream of character encodings. We can accurately model a single punch card, but when the card is reset to allow the user to work with a second card, the first is lost forever. As a card in this model can only encode 24 characters, any string longer than 24 characters cannot be represented by a single card and thus cannot be represented by the model. This limitation could be tackled in one of several ways. Perhaps most simply, we can extend the length of the card so that it contains 80 columns instead of 24 (as was the case for many later-generation punch cards, such as the IBM card format [Pugh, 1995]), but clearly this solution would face the same limitation for strings of over 80 characters. It should also be noted that in its current implementation, the time the model takes to initiate does increase linearly, dependent on the number of slots there are on the card. A more robust solution would be the ability to export used punch cards before starting on another, and import them back in at a later stage where they could be re-read with other cards to form a complete string. This functionality is limited by *Cadence*'s capabilities. A viable alternative to this solution would involve storing the 24-character string generated by each card, which can later be read back into the model and combined with all other strings to form a complete string. Whilst the card itself would be lost, the characters would not be, and indeed it would even be possible to reconstruct the punch card from each string of characters — almost the reverse process of what this model is trying to achieve.

As an emulation of real life punch cards, the model is realistic in that you cannot "undo" mistakes made in punching the card, as mentioned above. However, as a learning tool (which is one of the main areas of application for Empirical Modelling), it may be useful to have such functions available to the user, as re-punching an entire card due to one small mistake may be a tedious and unrewarding process. The user's time, for instance, may be more wisely spent by continuing to practice on various different strings rather than having to re-punch the same card multiple times. A possible extension of the model may be to implement an undo function that the user can choose to enable only if they wish to do so.

3 Evaluation

3.1 Of Empirical Modelling

The main principles of Empirical Modelling allow for the creation of high-level models that sufficiently mimic real life tools and their operations. As has been shown to be the case in this and previous studies, the "inner workings" of the models are not always true to the objects they are modelling. For instance, the system of assigning weights to the zone and numeric portions of the punch card, adding them, and finding the corresponding character in a lookup table, is not the method that would have been employed by punch card readers of previous generations.

In this sense, Empirical Modelling provides a "black box" environment, where focus to the user is on the end product rather than the inner workings of the model or mechanism. In the context of educational technology, such models create a very realistic and adaptable front-end environment for the user to interact with, but may fall down if the focus changes to describing *how* things work rather than just experimentation. This is not to say that designing a model with a realistic back-end using Empirical Modelling is impossible — in fact it is quite the opposite — but

this is restricted by the specific model in question, and the Empirical Modelling tools used to construct it.

3.2 Of *Cadence*

Cadence offers an interface that allows users from a wide range of backgrounds to create and develop an almost infinite range of potential models. Users of *Cadence* need not a detailed initial knowledge of modelling to get started. One of the main benefits that presented itself whilst developing in *Cadence* is the ability to start with a very basic model, and iteratively build upon it to create the final, more complex model. A similar benefit of the tool is how it presents an opportunity to extend and adapt models in ways that may be impossible, infeasible, or expensive to do in real life. This is much in line with the experimental aspect of Empirical Modelling.

4 Conclusions

The study has found that from a general perspective, Empirical Modelling can thoroughly and accurately represent dependencies between observables. In the field of educational technology, specifically in modelling mechanical devices such as the punch card, Empirical Modelling shows itself to be an adaptable and promising approach to computer-based modelling. A small number of weaknesses with regards to the approach have made themselves apparent during the study, namely the potential difficulty in creating a realistic model who's back-end remains true to the inner workings of the real life mechanism. This, however, is largely a limitation of the tools used within Empirical Modelling, rather than a fault with the concept itself. Empirical Modelling looks likely to continue to expand and gain recognition with the continued development of existing — and new — Empirical Modelling tools.

5 Acknowledgements

The author would like to thank Meurig Beynon, for providing what has proved to be an invaluable introduction to Empirical Modelling, and to Nick Pope, for development of the *Cadence* tool used in the study.

References

W M Beynon. Empirical Modelling for Educational Technology. *Cognitive Technology. 'Humanizing the Information Age'.*, pages 54–68, August 1997.

D Fisk. Programming with Punched Cards, 2005. URL `http://www.columbia.edu/cu/computinghistory/fisk.pdf`.

D W Jones. Punched Card Codes. URL `http://www.divms.uiowa.edu/ jones/cards/codes.html`.

R D Kloth. Card Punch Emulator, May 2003. URL `http://www.kloth.net/services/cardpunch.php`.

J Kopplin. An Illustrated History of Computers, Part 2, 2002. URL `http://www.computersciencelab.com/ComputerHistory/HistoryPt2.htm`.

E W Pugh. *Building IBM: Shaping an Industry and Its Technology*. The MIT Press, March 1995.

S Russ. Empirical Modelling: The Computer as a Modelling Medium. *Computer Bulletin*, pages 20–22, April 1997.

Student No. 0200114. An Investigation Into the Empirical Modelling of Physical Devices, in an Educational Context, Illustrated with the Enigma Machine Example. Technical report, University of Warwick, 2006.

Student No. 0215594. Modelling Babbage's Difference Engine. Technical report, University of Warwick, 2006.

L E Truedsell. *The Development of Punch Card Tabulation in the Bureau of the Census 1890-1940*, page 44. US GPO, 1965.

A S Weber. *Nineteeth Century Science: An Anthology*, chapter 9, page 84. Broadview Press, 2000.