FILOZOFIA I NAUKA
Studia filozoficzne i interdyscyplinarne
Tom 11, 2023

B. Jack Copeland, Diane Proudfoot

TURING'S WAGER?

doi: 10.37240/FiN. 2023.11.1.3

ABSTRACT

We examine Turing's intriguing claim, made in the philosophy journal *Mind*, that he had created a short computer program of such a nature that it would be impossible "to discover by observation sufficient about it to predict its future behaviour, and this within a reasonable time, say a thousand years" (Turing, 1950, p. 457). A program like this would naturally have cryptographic applications, and we explore how the program would most likely have functioned. Importantly, a myth has recently grown up around this program of Turing's, namely that it can be used as the basis of an argument—and was so used by Turing—to support the conclusion that it is impossible to infer a detailed mathematical description of the human brain within a practicable timescale. This alleged argument of Turing's has been dubbed "Turing's Wager" (Thwaites, Soltan, Wieser, Nimmo-Smith, 2017, p. 3) We demonstrate that this argument—in fact nowhere to be found in Turing's work—is worthless, since it commits a glaring logical fallacy. "Turing's Wager" gives no grounds for pessimism about the prospects for understanding and simulating the human brain.

Keywords: Alan Turing; Turing's Wager; mechanized encryption; laws of behaviour; unspecifiability of the mind; brain modelling; whole-brain simulation; cipher machines; Enigma; Fish; Tunny; early computer-based cryptography.

1. INTRODUCTION

We live in an age of misinformation. This article highlights a tiny, but important, piece of misinformation. A prominent online "List of things named after Alan Turing" includes—among many *bona fide* items such as "Turing test" and "Turing machine"—something called "Turing's Wager."[1] Turing's Wager has a *Wikipedia* entry (as well as a YouTube video "Turing's

[1] https://en.wikipedia.org/wiki/List_of_things_named_after_Alan_Turing

Wager—Know It ALL"[2]). This tells its readers that Turing's Wager is a philosophical argument "first given in 1950 by [...] Alan Turing in his paper *Computing Machinery and Intelligence*."[3] The conclusion of Turing's argument, readers are told, is that "it is impossible to infer or deduce a detailed mathematical model of the human brain within a reasonable timescale, and thus impossible in any practical sense." These claims are important because a number of large scientific research projects are currently trying to achieve exactly that, a detailed mathematical model of the whole human brain (for example the European *Human Brain Project*, Japan's *Brain/MINDS Project*, the *China Brain Project*, and the *BRAIN Initiative* in the United States). If this argument attributed to Turing is correct, then these whole-brain projects are wrong-headed and doomed to failure.

A recent article wields this claimed argument of Turing's against any empirical attempt to arrive at a whole-brain model (Thwaites, Soltan, Wieser, Nimmo-Smith, 2017). Its authors allege that Turing viewed the project of describing "the human brain in mathematical terms" with "blunt scepticism" (Thwaites et al., 2017, p. 1). They explain that "Turing's Wager (as we refer to it here) is an argument aiming to demonstrate that characterising the brain in mathematical terms will take over a thousand years," and they leave readers in no doubt that they are attributing this argument to Turing himself: "Turing introduced [...] *Turing's Wager* in [his] essay, *Computing Machinery and Intelligence*' (Thwaites et al., 2017, p. 3).

Their statement of the wager argument is concise. They explain that it utilizes a concrete illustration, by means of which Turing "sought to highlight the challenges involved." This is a "short computer program" that he wrote for the University of Manchester computer; it "accepted a single number, performed a series of unspecified calculations on it, and returned a second number" (Thwaites et al., 2017, p. 1).

> "It would be extremely difficult, Turing argued, for anyone to guess these calculations from the input and output numbers alone. Determining the calculations taking place in the brain, he reasoned, must be harder still: not only does the brain accept tens-of-thousands of inputs from sensory receptors around the body, but the calculations these inputs undergo are far more complicated than anything written by a single programmer. Turing underscored his argument with a wager: that it would take an investigator at least a thousand years to guess the full set of calculations his Manchester program employed. Guessing the full set of calculations taking place in the brain, he noted, would appear prohibitively time-consuming (Turing 1950)" (Thwaites et al., 2017, pp. 1–2).

---

[2] https://www.youtube.com/watch?v=ONxwksicpV8
[3] https://en.wikipedia.org/wiki/Turing%27s_Wager

Our aims are (1) to publicize the fact that there is nothing like this argument to be found in Turing's article (nor elsewhere in his writings); (2) to clarify the nature and purpose of Turing's—very interesting—short program, around which Thwaites et al. have arranged their wager argument; and (3) to assess the implications for the study of the brain of Turing's actual claims about this program.

The first appearance in the historical record of Turing's short program occurs in some notes (Copeland, 2005) which were taken during a discussion on an autumn evening in 1949, in Manchester, where Turing was directing the university's Computing Machine Laboratory. In this discussion, Turing mentioned a program whose nature it would be "impossible" to deduce from observations of its input–output behaviour. He used this example to defeat an argument against the possibility of machine intelligence. Yet he gave few clues as to how the program worked. What was its structure such that it could defy analysis for (he said) "a thousand years?" Our suggestion will be that the program simulated a type of cipher device, and was perhaps connected with Turing's post-war work for GCHQ (the UK equivalent of the US National Security Agency). After our efforts to piece together the textual clues concerning Turing's mysterious program, we will go on to investigate the textual evidence, or lack of it, for the Thwaites et al. interpretation of Turing's position.

2. THE 1949 MANCHESTER DISCUSSION

In the notetaker's record of that 1949 discussion, held at Manchester University on 27 October, Turing is reported as making the intriguing claim that, in certain circumstances, "it would be impossible to find the programme inserted into quite a simple machine." That is to say, for the machine and program Turing was considering, reverse-engineering the program from the machine's behaviour is in practice not possible.

The discussion involved Michael Polanyi, Dorothy Emmet, Max Newman, Geoffrey Jefferson, J. Z. Young, and others (the notetaker was Wolfe Mays). At that particular point in the discussion, Turing was responding to Polanyi's assertion that "a machine is fully specifiable, while a mind is not." The mind is "only said to be unspecifiable because it has *not yet been* specified," Turing replied; and it does not follow from this, he said, that "the mind is unspecifiable"—any more than it follows from the inability of investigators to specify the program in his "simple machine" that this program is unspecifiable. After all, Turing knew the program's specification.

Polanyi's assertion is not unfamiliar; other philosophers and scientists make claims in a similar spirit. Recent examples are "mysterianist" philosophers of mind, who claim that the mind is "an ultimate mystery, a mystery

that human intelligence will never unravel" (McGinn, 1999, p. 5).[4] So, what was Turing's machine, such that it might counterexample a claim like Polanyi's? A machine that—although "quite a simple" one—thwarted attempts to analyze it?

3. BACKGROUND: TURING'S KNOWLEDGE OF CIPHER MACHINES

The cipher machines used in the Second World War could certainly be described as "simple," despite the monumental difficulty of inferring their mode of operation from their input–output behaviour. These machines typically employed a system of rotating code-wheels to encrypt a message. The best-known example is the German Enigma machine, with its three code-wheels. Some later Enigma models contained a fourth wheel, with a consequent step-change in the level of security the machine could provide.

The form of Enigma used by the German military was derived from an earlier commercial version of the machine, by a series of significant enhancements that greatly increased security. In Germany, the military machine had the reputation of being well-nigh unbreakable; yet from 1933, Polish codebreakers regularly decrypted Enigma messages intercepted from the German Army's radio network (Copeland, 2004). The Polish codebreakers, led by mathematician Marian Rejewski, also broke into the Enigma traffic of the German Air Force and Navy. Statistics gathered by the Biuro Szyfrów—the Polish Cipher Bureau—in 1938 showed that the Poles were by then successfully decrypting about 75 per cent of all intercepted Enigma material.

It was around that time that Rejewski, together with the engineer Antoni Palluth, designed the *bomba*, an ingenious machine for breaking Enigma messages. The bomba contained eighteen rotating wheels, each one simulating an Enigma wheel; thus, the bomba's wheels collectively simulated six three-wheel Enigma machines. By mid-November 1938, half a dozen bomby were in continuous operation at an underground facility near Warsaw. British and French codebreakers were invited to view the bomby, as well as other items of codebreaking technology, in the summer of 1939. When Turing joined Bletchley Park—the British wartime headquarters for military codebreaking—in the autumn of that year, the principles of the bomba were explained to him, and he went on to design his famous bombe, based on the bomba but larger, containing more than a hundred rotating drums. Like the bomba's wheels, each drum simulated a single Enigma wheel. It was initially thought the bombe would be able to use the same codebreaking method that

[4] McGinn is here describing, not only the mind, but the "bond between the mind and the brain."

the bomba mechanized, until this method became wholly ineffective, due to a sudden change in German operating procedures in the spring of 1940. The actual bombe mechanised a very different codebreaking method, devised by Turing.[5]

The Enigma machine had operational drawbacks—it was slow to use, as well as labour-intensive (requiring three operators at each end of the sender–receiver link), and moreover the practical upper limit on message size was only a few hundred characters. From 1940, the German military began to roll out a new breed of cipher machine (Copeland, 2006). These were collectively termed "Fish" at Bletchley Park. The British knew of three types of Fish machine; they named them "Tunny," "Sturgeon" and "Thrasher." The first experimental Tunny radio link went into operation in June 1941, and soon the Tunny machine was being used for the highest-level German Army message-traffic, between Berlin and the generals and field-marshals in charge of the fighting at the various battle-fronts. Tunny had twelve code-wheels, Sturgeon had ten. The shadowy Thrasher remained unbroken; it seems to have used a pseudo-random tape produced by a wheeled machine. Bletchley Park broke both Tunny and Sturgeon. Tunny, in particular, produced a deluge of intelligence. Breaking into the Tunny system was a team effort and Turing played a fundamental role (in 1942).

Wheeled cipher machines remained the principal means of encryption during the post-war years. Enigma was adopted by several Warsaw Pact countries, including East Germany, where the *Stasi* used it during the 1940s and 1950s (Weierud, 2006). It was also used by the Norwegian Security Police, until the 1960s. Large numbers of Sturgeon machines were employed by the French, Dutch, Norwegians, Swedes, and others. Tunny, though, was arguably the most important of the wheeled cipher-machines. The method of encryption it pioneered was a staple of military and commercial cryptosystems for many decades after the war.

4. MORE INFORMATION ABOUT TURING'S "SIMPLE MACHINE"

Turing fleshed out his example a little in his 1950 article *Computing Machinery and Intelligence* (Turing, 1950). He was arguing against the proposition that "given a discrete-state machine it should certainly be possible to discover by observation sufficient about it to predict its future behaviour, and this within a reasonable time, say a thousand years" (Turing, 1950,

[5] Rejewski's method attacked what was called the message's *indicator*. This consisted of enciphered information about how the sender's machine was configured, and the indicator was transmitted to the receiver immediately prior to the message itself. Turing's method, on the other hand, attacked the message text directly, by means of what he called "closures" in the relationship between the enciphered characters and their unenciphered equivalents.

p. 457). This "does not seem to be the case," he said, and he went on to describe a counterexample:

> "I have set up on the Manchester computer a small programme using only 1000 units of storage, whereby the machine supplied with one sixteen figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values" (Turing, 1950, p. 457).

These passages occur in a short section titled "The Argument from Informality of Behaviour," in which Turing's aim was to refute an argument purporting to show that "we cannot be machines" (1950, p. 457). The argument, as Turing explained it, is this:

(1) If each man had a definite set of laws of behaviour which regulate his life, he would be no better than a machine.
(2) But there are no such laws.
(3) Men cannot be machines.[6]

Turing agreed that "being regulated by laws of behaviour implies being some sort of machine (though not necessarily a discrete-state machine)," and that "conversely being such a machine implies being regulated by such laws" (1950, p. 457). If this biconditional serves as a reformulation of the argument's first premiss, then the argument is plainly valid.

Turing's strategy was to challenge the argument's second premiss. He said:

> "... we cannot so easily convince ourselves of the absence of complete laws of behaviour [...] The only way we know of for finding such laws is scientific observation, and we certainly know of no circumstances under which we could say 'We have searched enough. There are no such laws'" (1950, p. 457).

Turing then offered his example of the "small programme" that cannot be reverse-engineered, in order to demonstrate "more forcibly" that the failure to find laws of behaviour does not imply that no such laws are in operation (Turing, 1950, p. 457).

These are the only appearances of Turing's "simple machine" in the historical record (at any rate, in the declassified record). How could Turing's mysterious program have worked, such that in practice it defied analysis? And what implications might the program have for the study of the brain—beyond Turing's uses of it against Polanyi's bold assertion and against the "informality of behaviour" argument? We discuss these questions in turn.

---

[6] Turing first stated the argument in this form: "If each man had a definite set of rules of conduct by which he regulated his life he would be no better than a machine. But there are no such rules, so men cannot be machines" (1950, p. 457). He then considered the argument that results if "we substitute 'laws of behaviour which regulate his life' for 'laws of conduct by which he regulates his life' " (ibidem).

One obvious point about Turing's mysterious program (henceforward: XX) is that it amply meets the specifications for a high-grade cipher machine. It is seldom noted that Turing's career as a cryptographer did not end with the defeat of Hitler. During the post-war years, as well as playing a leading role in the Manchester Computing Machine Laboratory, Turing worked as a consultant for GCHQ, Bletchley Park's peacetime successor (Copeland, 2017, p. 37). With the development of the first all-purpose electronic computers, two of Turing's great passions, computing and cryptography, were coalescing. He was an early pioneer in the application of electronic stored-program computers to cryptography.

The Manchester computer's role in Cold War cryptography remains largely classified. We know, however, that while the computer was at the design stage, Turing and his Manchester colleague Max Newman—both had worked on breaking the Tunny cipher system at Bletchley Park—directed the engineers to include special facilities for cryptological work.[7] These included operations for differencing, a now familiar cryptological technique that originated in Turing's wartime attack on Tunny, and was known at Bletchley Park as "delta-ing." GCHQ itself took a keen interest in the Manchester computer. Jack Good, who in 1947 had a hand in the design of Manchester's prototype "Baby" computer, joined GCHQ full-time in 1948 (Copeland, 2011, pp. 5–6, 28–29). Others at Manchester who were closely involved with the computer also consulted for GCHQ (Copeland, 2011, p. 6); and a contingent from GCHQ attended the inaugural celebration for what Turing called the Mark II[8] version of the Manchester computer, installed in Turing's lab in 1951. The idea of programming electronic digital computers to encrypt military and commercial material was new and promising. GCHQ installed a Mark II at its new headquarters in Cheltenham.[9]

5. XX AS AN ENCRYPTION DEVICE

How might XX be used for encryption? A hypothetical example illustrates the general principles. Suppose Alice wishes to encipher her message "I LUV U" (the "plaintext") before sending the result (the "ciphertext") to Bob. Bob, who knows Alice's enciphering method, will uncover the plaintext by using Alice's method in reverse.

Alice's first step is to convert the plaintext into binary. Turing would have done this using teleprinter code (also known as Baudot-Murray code). Em-

---

[7] Tom Kilburn in interview with Copeland, July 1997; G. C. Tootill, *Informal Report on the Design of the Ferranti Mark I Computing Machine*, November 1949, National Archive for the History of Computing, University of Manchester, p. 1.

[8] The computer that Turing called the Mark II is also known as the Ferranti Mark I, after the Manchester engineering firm that built it.

[9] The manufacturer's name for the model installed at GCHQ was the Ferranti Mark I Star.

ployed worldwide in communications systems at that time, teleprinter code transformed each keyboard character into a different string of five bits; for example, "A" was 11000 and "B" was 10011. Teleprinter code is the ancestor of the ASCII and UTF-8 codes used today to represent text digitally. Turing was very familiar with teleprinter code from his time at Bletchley Park, since the German Tunny system used it. In fact, Turing liked teleprinter code so much that he chose it as the basis for the Manchester computer's programming language.

To convert the plaintext into binary, Alice needs to know the following teleprinter code equivalences: "I" is 01101; "L" is 01001; "U" is 11100; "V" is 01111; and space is 00100. To do the conversion, she first writes down the teleprinter code equivalent of "I," and then (writing from left to right) the teleprinter code equivalent of space, and then of "L," and so on, producing:

01101001000100111100011110010011100

This string of 35 figures (or bits) is called the "binary plaintext."

So far, there has been no encryption, only preparation. The encryption will be done by XX. Recall that XX takes a sixteen-figure number as input and responds with another sixteen-figure number. Alice readies the binary plaintext for encryption by splitting it into two blocks of sixteen figures, with three figures "left over" on the right:

0110100100010011 1100011110010011 100

Next, she pads out the three left-over figures so as to make a third sixteen-figure block. To do this, she first adds "/" (00000), twice, at the end of the binary plaintext, so swelling the third block to thirteen figures, and then she adds (again on the far right of the third block) three more bits, which she selects at random (say 110), so taking the number of figures in the third block to sixteen. The resulting three blocks form the "padded binary plaintext":

0110100100010011 1100011110010011 1000000000000110

Alice now uses XX to encrypt the padded binary plaintext. She inputs the left-hand sixteen-figure block and writes down XX's sixteen-figure response; these are the first sixteen figures of the ciphertext. Then she inputs the middle block, producing the next sixteen figures of the ciphertext, and then the third block. Finally, she sends the ciphertext, 48 figures long, to Bob. Bob splits up the 48 figures of ciphertext into three sixteen-figure blocks and decrypts each block using his own XX (configured identically to Alice's); and then, working from the left, he replaces the ensuing five-figure groups with their teleprinter code equivalent characters. He knows to discard any terminal occurrences of "/," and also any group of fewer than five figures following the trailing "/." Bob is now in possession of Alice's plaintext.

This example illustrates how XX could have been used for cryptography; but it gets us no closer, however, to knowing how XX generated its sixteen-figure output from its input. Probably this will never be known—unless the classified historical record happens to include information about XX, but this seems unlikely. However, let us speculate. As previously mentioned, the leading cipher machines of that era—Enigma, Tunny, and Sturgeon, as well as the Hagelin, the British Typex and Portex, and Japanese machines such as Purple—all used a system of wheels to produce the ciphertext from the plaintext. We shall focus on Tunny, since it is the simplest of these machines to describe, and also because of its importance post-war. At Bletchley Park, Turing had invented the first systematic method for breaking the German Army's Tunny messages; it is quite possible that he was interested after the war in refining the machine's principles of encryption for future applications.

6. SIMULATING CODE-WHEEL CIPHER-MACHINES

The Tunny machine had at its heart twelve code-wheels, but here we shall focus on a form of the Tunny machine with only ten code-wheels. Turing's wartime Tunny-breaking colleagues Jack Good and Donald Michie have argued persuasively that if (counterfactually) the Germans had used this ten-wheel version of the machine, it would have offered a far higher level of crypto-security than the twelve-wheel machine (Good, Michie, 2006). In fact, Michie remarked that, had the Germans used the ten-wheel version, "it is overwhelmingly probable that Tunny would never have been broken." With the ten-wheel machine, he said, there would be no "practical possibility of reverse-engineering the mechanism that generated it" (Good, Michie, 2006, p. 409). Assuming that the machine was not compromised by security errors, and that the state of the art in cryptanalysis persisted much as it was in 1949, then the ten-wheel Tunny might indeed have remained unbroken for Turing's "a thousand years." If Turing was interested in Tunny post-war, it was most probably in this form of the machine.

As far as the user is concerned, the Tunny machine (both the ten- and twelve-wheel versions) is functionally similar to XX. When supplied with one five-figure number, the Tunny machine responds with another. When the number that is supplied (either by keyboard or from punched paper tape) is the teleprinter code of a letter of plaintext, the machine's reply provides the corresponding five figures of ciphertext. If, on the other hand, the machine is being used, not to encrypt the plaintext, but to decrypt the ciphertext, then its reply to five figures of ciphertext is the teleprinter code of the corresponding plaintext letter.

The machine produces its reply by first generating five figures internally, and then "adding" these to the number that is supplied as input. Tunny "ad-

dition" is better known to logicians as exclusive disjunction: 0 + 0 = 0, 1 + 0 = 1, 0 + 1 = 1, and 1 + 1 = 0. For example, if the incoming five figures are 01101, and the internally generated five figures are 00100, then the machine's reply is 01001 (i.e. 01101 + 00100).

The function of the code-wheels is to generate the five figures that are added to the incoming number. A simple way to generate five figures is to use an arrangement of five wheels, each of which contributes one figure. However, the set-up actually used in the ten-wheel Tunny machine (and the same in the twelve-wheel version) is more complicated, the aim being greater security. Rather than a single group of five wheels, there are two groups, with five wheels in each group. In Bletchley Park jargon, the two groups were known respectively as the "Φ-wheels" and the "Ψ-wheels." Each group of wheels produces five figures; and these two five-figure numbers are then added together. It is the result of this addition that the machine goes on to add to the incoming number.

The Tunny machine's action is transparently described by the machine's so-called "encipherment equation:"

$$(\Phi + \Psi) + P = C$$

Adding the number Φ that is produced by the Φ-wheels to the number Ψ produced by the Ψ-wheels, and then adding the resulting number to P—the incoming five figures of binary plaintext—produces C, the corresponding five figures of ciphertext. With each incoming five-figure number, every wheel of the 10-wheel machine turns forwards a step; this has the result that the internally-generated number Φ + Ψ is always changing. (Incidentally, the function of the twelve-wheel Tunny's two extra wheels was quite different. Known as the "motor wheels," these served to create irregularities in the motions of the Ψ-wheels. No doubt the engineers at Lorenz[10] thought this arrangement would enhance the security of the machine, but they were badly mistaken. The motor wheels introduced a serious weakness, and this became the basis of Bletchley Park's highly successful attack on the twelve-wheel Tunny machine.)

One last relevant detail about Tunny's wheels. Each wheel had pins spaced regularly around its circumference. An operator could set each pin into one of two different positions, protruding or not protruding. (For security, the positions were modified daily.[11]) An electrical contact read figures from the rotating wheel (one contact per wheel): a pin in the protruding position would touch the contact, producing 1 (represented by electricity flowing), while a non-protruding pin would miss the contact, producing 0

[10] The Tunny machine was manufactured by the Berlin engineering company C. Lorenz AG. For that reason it was also called the "Lorenz machine" at post-war GCHQ (although never at wartime Bletchley Park, where the manufacturer was unknown and the British codename "Tunny machine" was invariably used).

[11] From 1 August 1944.

(no flow). As a group of five wheels stepped round, the row of five contacts delivered five-figure numbers. Each wheel had a different number of pins, ranging from 23 to 61; at Bletchley Park, this number was referred to as the length of the wheel.

It would have been completely obvious to the post-war pioneers of computerized cryptography that one way to create a secure enciphering program was to simulate an existing secure machine. Turing's mysterious program may well have been a simulation of the ten-wheel Tunny machine, or of some other wheeled cipher machine.

Turing's brief descriptions of XX contain some small numerical clues. He gave in effect an upper bound on the number of instruction-executions that were performed in the course of encrypting one sixteen-figure number: XX gives its reply "within two seconds," he said. In 1949-1950, most of the Manchester computer's instructions took 1.8 milliseconds to execute; so approximately 1000 instructions could be implemented in two seconds. He also said that XX required 1000 units of storage. In the Manchester computer as it was in 1949–1950, a unit of high-speed storage consisted of a line of 40 bits spread horizontally across the screen of a Williams tube (Turing c.1950).[12] (A Williams tube, the basis of the computer's high-speed memory, was a cathode ray tube; a small dot of light on the tube's screen represented 1 and a large dot 0.) 1000 units is therefore 40,000 bits of storage. To simulate the ten-wheel Tunny on the Manchester computer, Turing would have needed ten variable-length shift registers to represent the wheels. Since the lengths of the ten wheels were, respectively, 41, 31, 29, 26, 23, 43, 47, 51, 53, and 59, a total of 403 bits of storage would be required for the pin patterns. This leaves more than 39 kilobits, an ample amount for storing the instructions—which add Φ, Ψ and P, shift the bits in the wheel registers (simulating rotation), and perform sundry control functions—and for executing them.

Why, one might wonder, are the numbers encrypted by XX sixteen figures long? This might indicate that XX simulated a cipher machine with more than ten wheels—possibly a Tunny-like machine with modifications introduced by Turing for greater security: he might have increased the number of Φ-wheels and Ψ-wheels (and also the lengths of the wheels), or made other modifications that are impossible now to reconstruct. On the other hand, however, the number sixteen might in fact be no guide at all to the number of wheels. During 1941, when Tunny was first used for military traffic, it was relatively obvious to the Bletchley Park codebreakers that the new machine had twelve wheels—invaluable information. Turing's choice of sixteen-figure numbers (rather than some number of figures bearing an immediate relationship to the number of wheels) might simply have been a way of masking the number of wheels.

---

[12] Turing described the computer as it was at that time in an Appendix to (Turing, c.1950) entitled "The Pilot Machine (Manchester Computer Mark I)."

Our first question about Turing's mysterious program was: how could it have worked, such that in practice it defied analysis? One plausible answer is: by simulating a Tunny or other wheeled cipher-machine. We turn now to the question: does XX have implications concerning the feasibility of whole-brain simulation?

7. XX AND BRAIN SIMULATION

According to Thwaites, Soltan, Wieser and Nimmo-Smith, the answer to that question is a resounding *yes*. As we mentioned earlier, they claim to find in Turing "an argument aiming to demonstrate that characterising the brain in mathematical terms will take over a thousand years" (Thwaites et al., 2017, p. 3), and they say that XX serves as a practical illustration of the challenges involved.

However, the only conclusion that Turing drew from the XX example was (as we described above) that failing to find the laws of behaviour or a full specification does not imply that none exist. Contrary to what Thwaites and his co-authors say, there is no argument in "Computing Machinery and Intelligence" (nor elsewhere in Turing's writings) aiming to demonstrate that "characterising the brain in mathematical terms will take over a thousand years." It is false that Turing noted (as Thwaites and his co-authors claim) anything to the effect that "[g]uessing the full set of calculations taking place in the brain would appear prohibitively time-consuming," or that he reasoned in "Computing Machinery and Intelligence" about the difficulty of determining "the calculations taking place in the brain" (Thwaites et al., 2017, pp. 1, 2). Thwaites and his co-authors tell us that Turing was not "optimistic about [the] chances of beating Turing's Wager" (Thwaites et al., 2017, p. 3), but this is an extraordinary claim—Turing never mentioned the so-called Wager.

A defender of Turing's Wager might perhaps respond that the fact that Turing himself did not state or suggest "Turing's Wager" is of only historical or scholarly importance. If valid, the wager argument is certainly significant, owing to its powerful negative implications about the feasibility of whole-brain simulation, as discussed above. But is the wager argument valid?

Set out explicitly, the wager argument is as follows:

(1) It would take at least 1000 years to determine the calculations occurring in XX.
(2) The calculations occurring in the brain are far more complicated than those occurring in XX.
∴ (3) It would take well over 1000 years to determine the calculations occurring in the brain.

Both (1) and (2) are true, we may assume—certainly the calculations done by the ten-wheel Tunny are extremely simple in comparison with those taking place in the brain. However, these premises do *not* entail (3). If XX simulates a cryptographic machine, something carefully and cleverly designed to thwart any efforts to determine the calculations taking place within it, there is no reason why a more complicated but potentially more transparent machine should not succumb to analysis more quickly than XX. The mere possibility that XX simulates a cipher-machine, a machine designed to be unfathomable, shows that in some possible world (1), (2), and the negation of (3) are true, and thus that the Turing's wager argument is invalid.

The answer to our second question, then, is *no*: XX has nothing to tell us about the prospects of whole-brain simulation.

8. CONCLUSION

In the 1949 Manchester discussion, Turing employed one of his hallmark techniques: attacking a grand thesis with a concrete counterexample. He used XX to undermine both Polanyi's claim that "a machine is fully specifiable, while a mind is not" and the Informality of Behaviour Argument against machine intelligence. But his writings contain no trace of an attempt to use XX to undermine whole-brain modelling. "Turing's Wager" is a fabrication, as is the claim by Thwaites and his co-authors that Turing "noted" that a quest for "the full set of calculations taking place in the brain [...] would appear prohibitively time-consuming." Moreover, as we have just argued, the attempt by Thwaites et al. to recruit XX to their effort to undermine the viability of whole-brain modelling is deeply misguided.

Although Turing himself made no connection between XX and the prospects for brain-modelling, one may still ask: What might Turing have thought of the *BRAIN Initiative* and other large-scale brain-modelling projects? It is impossible to say—but Turing was, after all, an early pioneer of brain-modelling. Not long after the war, he wrote:

> "In working on the ACE I am more interested in the possibility of producing models of the action of the brain than in the practical applications to computing. [...] [A]lthough the brain may in fact operate by changing its neuron circuits by the growth of axons and dendrites, we could nevertheless make a model, within the ACE, in which this possibility was allowed for, but in which the actual construction of the ACE did not alter."[13]

Turing might well have cheered on his 21st century descendants.[14]

---

[13] Letter from Turing to W. Ross Ashby, undated, circa 1947 (Woodger Papers, Science Museum, London, catalogue reference M11/99). Cf. *The Essential Turing*, B. J. Copeland (Ed.), Oxford University Press, 2004, p. 375.

[14] This article is a derivative of one that appeared in the 2019 APA Newsletter on Philosophy and Computers.

REFERENCES

Copeland, B. J., *Enigma*, in: *The Essential Turing*, B. J. Copeland (ed.), Oxford University Press, Oxford–New York 2004.

_____ , *The German Tunny Machine*, in: *Colossus: The Secrets of Bletchley Park's Codebreaking Computers*, B. J. Copeland et al., Oxford University Press, Oxford–New York 2006.

_____ , *The Manchester Computer: A Revised History*, IEEE Annals of the History of Computing, 2011, 33, pp. 4–37.

_____ , *Crime and Punishment*, in: Copeland, B. J., J. Bowen, M. Sprevak, R. Wilson, et al., *The Turing Guide*, Oxford University Press, Oxford–New York 2017.

Copeland, B. J. (ed.), *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life*, Oxford University Press, Oxford–New York 2004.

Copeland, B. J. (ed.), *"The Mind and the Computing Machine," by Alan Turing and Others*, The Rutherford Journal: The New Zealand Journal for the History and Philosophy of Science and Technology, vol. 1, 2005; http://www.rutherfordjournal.org/ article010111.html

Copeland, B. J., et al., *Colossus: The Secrets of Bletchley Park's Codebreaking Computers*, Oxford University Press, Oxford–New York 2006.

Copeland, B. J., Bowen, J., Sprevak, M., Wilson, R. et al., *The Turing Guide*, Oxford University Press, Oxford–New York 2017.

Good, I. J., Michie, D., *Motorless Tunny*, in: Copeland, B. J., et al., *Colossus: The Secrets of Bletchley Park's Codebreaking Computers*, Oxford University Press, Oxford–New York 2006.

McGinn, C., *The Mysterious Flame: Conscious Minds in a Material World*, New York: Basic Books, 1999.

Thwaites, A., Soltan, A., Wieser, E., Nimmo-Smith, I., *The Difficult Legacy of Turing's Wager*, Journal of Computational Neuroscience, 2017, 43, pp. 1–4.

Turing, A. M., *Programmers' Handbook for Manchester Electronic Computer Mark II*. Computing Machine Laboratory, University of Manchester; no date, c. 1950; http://www.alanturing.net/turing_archive/archive/m/m01/M01-001.html

_____ , *Computing Machinery and Intelligence* (1950), in: *The Essential Turing*, B. J. Copeland (ed.), Oxford University Press, Oxford–New York 2004.

Weierud, F., *Bletchley Park's Sturgeon—the Fish that Laid No Eggs*, in: Copeland, B. J., et al., *Colossus: The Secrets of Bletchley Park's Codebreaking Computers*, Oxford University Press, Oxford–New York 2006.

ABOUT THE AUTHORS:

B. Jack Copeland — FRS NZ: Distinguished Professor of Philosophy, University of Canterbury, New Zealand.

Email: jack.copeland@canterbury.ac.nz

Diane Proudfoot — Professor of Philosophy, University of Canterbury, New Zealand.

Email: diane.proudfoot@canterbury.ac.nz