# 3 Hello, AI

We've covered hardware, software, and programming. It's time to move on to a more advanced programming topic: artificial intelligence. To most people, the phrase *artificial intelligence* suggests something cinematic—maybe Commander Data, the lifelike cyborg from *Star Trek: The Next Generation*; perhaps Hal 9000 from *2001: A Space Odyssey*; or Samantha, the AI system from the movie *Her*; or Jarvis, the AI majordomo that helps Iron Man in the Marvel comics and movies. Regardless, here's what's important to remember: *those are imaginary*. It's easy to confuse what we imagine and what is real—especially when we want something very badly. Many people want AI to be real. This usually takes the form of wanting a robot butler to attend to your every need. (I will confess to having had many late-night undergraduate conversations about the practical and ethical considerations of having a robot butler.) A disproportionate number of the people who make tech fall into the camp of desperately wanting Hollywood robots to be real. When Facebook's Mark Zuckerberg built an AI-based home automation system, he named it Jarvis.

One excellent illustration of the confusion between real and imaginary AI happened to me at the NYC Media Lab's annual symposium, a kind of science fair for grownups. I was giving a demo of an AI system I built. I had a table with a monitor and a laptop hooked up to show my demo; three feet away was another table with another demo by an art school undergraduate who had created a data visualization. Things became boring when the crowd died down, so we got to chatting.

"What's your project?" he asked.

"It's an artificial intelligence tool to help journalists quickly and efficiently uncover new story ideas in campaign finance data," I said.

"Wow, AI," he said. "Is it a *real* AI?"

"Of course," I said. I was a little offended. I thought: *Why would I spend my day demonstrating software at this table if I hadn't made something that worked?*

The student came over to my table and started looking closely at the laptop hooked up to the monitor. "How does it work?" he asked. I gave him the three-sentence explanation (you'll read the longer explanation in chapter 11). He looked confused and a little disappointed.

"So, it's not real AI?" he asked.

"Oh, it's real," I said. "And it's spectacular. But you know, don't you, that there's no simulated person inside the machine? Nothing like that exists. It's computationally impossible."

His face fell. "I thought that's what AI meant," he said. "I heard about IBM Watson, and the computer that beat the champion at Go, and self-driving cars. I thought they invented real AI." He looked depressed. I realized he'd been looking at the laptop because he thought there was something in there—a "real" ghost in the machine. I felt terrible for having burst his bubble, so I steered the conversation toward a neutral topic—an upcoming Star Wars movie—to cheer him up.

This interaction stuck with me because it helps me remember the difference between how computer scientists think about AI and how members of the public—including highly informed undergraduates working on tech—think about AI.

*General AI* is the Hollywood kind of AI. General AI is anything to do with sentient robots (who may or may not want to take over the world), consciousness inside computers, eternal life, or machines that "think" like humans. *Narrow AI* is different: it's a mathematical method for prediction. There's a lot of confusion between the two, even among people who make technological systems. Again, general AI is what some people want, and narrow AI is what we have.

One way to understand narrow AI is this: narrow AI can give you the most likely answer to any question that can be answered with a number. It involves quantitative prediction. Narrow AI is statistics on steroids.

Narrow AI works by analyzing an existing dataset, identifying patterns and probabilities in that dataset, and codifying these patterns and probabilities into a computational construct called a *model*. The model is a kind of black box that we can feed data into and get an answer out of. We can take the model and run new data through it to get a numerical answer that

predicts something: how likely it is that a squiggle on a page is the letter *A*; how likely it is that a given customer will pay back the mortgage money a bank loans to him; which is the best next move to make in a game of tic-tac-toe, checkers, or chess. Machine learning, deep learning, neural networks, and predictive analytics are some of the narrow AI concepts that are currently popular. For every AI system that exists today, there is a logical explanation for how it works. Understanding the computational logic can demystify AI, just like dismantling a computer helps to demystify hardware.

AI is tied up with games—not because there's anything innate about the connection between games and intelligence, but because computer scientists tend to like certain kinds of games and puzzles. Chess, for example, is quite popular in their crowd, as are strategy games like Go and backgammon. A quick look at the Wikipedia pages for prominent venture capitalists and tech titans reveals that most of them were childhood Dungeons & Dragons enthusiasts.

Ever since Alan Turing's 1950 paper that proposed the *Turing test* for machines that think, computer scientists have used chess as a marker for "intelligence" in machines. Half a century has been spent trying to make a machine that could beat a human chess master. Finally, IBM's Deep Blue defeated chess champion Garry Kasparov in 1997. AlphaGo, the AI program that won three of three games against Go world champion Ke Jie in 2017, is often cited as an example of a program that proves general AI is just a few years in the future. Looking closely at the program and its cultural context reveals a different story, however.

AlphaGo is a human-constructed program running on top of hardware, just like the "Hello, world" program you wrote in chapter two. Its developers explain how it works in a 2016 paper published in *Nature*, the international journal of science.[1] The opening lines of the paper read: "All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state $s$, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately $b^d$ possible sequences of moves, where $b$ is the game's breadth (number of legal moves per position) and $d$ is its depth (game length)." This is perfectly clear to someone who has years of high-level mathematical training, but many of us would prefer a plainer-language explanation.

To understand AlphaGo, it helps to start by thinking about tic-tac-toe, a game that most children have mastered. If you go first in tic-tac-toe and choose the space in the middle of the nine-square grid, you can always play to a win or a draw. Going first gives you an advantage: you will have five moves to your opponent's four. Most kids grasp this intuitively and insist on going first when playing with an indulgent older opponent.

It's also relatively easy to write a computer program to play tic-tac-toe against a human opponent. The first one was written in 1952. There's an *algorithm*, a set of rules or steps, that you can deploy so that the computer always plays to a win or a draw. Like "Hello, world," building a tic-tac-toe game is a common exercise in introductory computing classes.

Go is far more sophisticated than tic-tac-toe, but it's also a game played on a grid. Each Go player receives a pile of either black or white stones. Beginners play on a grid made of nine vertical and nine horizontal lines; advanced players use a nineteen-by-nineteen grid. Black goes first and places a black stone at an intersection of two lines. White then places her stone at a different intersection. The players alternate turns, with the goal of "capturing" the opponent's stones by surrounding a stone with the opposite color.

People have been playing Go for three thousand years. Computer scientists and Go aficionados have been studying patterns in the game since at least 1965. The first computerized Go program was written in 1968. There's an entire subfield of computer science research devoted to Go, called (unsurprisingly) *Computer Go*.

For years, Computer Go players and researchers have been amassing records of games. A game record looks like this:

```
(;GM[1]
FF[4]
SZ[19]
PW[Sadavir]
WR[7d]
PB[tzbk]
BR[6d]
DT[2017-05-01]
PC[The KGS Go Server at http://www.gokgs.com/]
KM[0.50]
RE[B+Resign]
RU[Japanese]
```

```
CA[UTF-8]
ST[2]
AP[CGoban:3]
TM[300]
OT[3x30 byo-yomi]
;B[qd];W[dc];B[eq];W[pp];B[de];W[ce];B[dd];W[cd];B[ec];W[cc];B
[df];W[cg];B[kc];W[pg];B[pj];W[oe];B[oc];W[qm];B[of];W[pf];B[p
e];W[og];B[nf];W[ng];B[nj];W[lg];B[mf];W[lf];B[mg];W[mh];B[me]
;W[li];B[kh];W[lh];B[om];W[lk];B[qo];W[po];B[qn];W[pn];B[pm];W
[ql];B[rq];W[qq];B[rm];W[rl];B[rn];W[rj];B[qr];W[pr];B[rr];W[m
n];B[qi];W[rh];B[no];W[on];B[nn];W[nm];B[nl];W[mm];B[ol];W[mp]
;B[ml];W[ll];B[np];W[nq];B[mo];W[mq];B[lo];W[kn];B[ri];W[si];B
[qj];W[qk];B[kq];W[kp];B[ko];W[jp];B[lp];W[lq];B[jq];W[jo];B[j
n];W[in];B[lm];W[jm];B[ln];W[hq];B[qh];W[rg];B[nh];W[re];B[rd]
;W[qe];B[pd];W[le];B[md])
```

The text may look like gobbledygook to a human, but it's highly structured so that a machine can process it easily. The structure is called *smart games format* (SGF). The text shows who played the game, where, what each of the moves were, and how the game was resolved.

The large text area shows all the moves. Columns in the Go grid are labeled in alphabetical order from left to right, and rows are labeled from top to bottom. In this game, Black (B) went first and placed a stone at the intersection of column q and row d. This is shown as `;B[qd]`. Then, the text `;W[dc]` shows that White (W) placed a stone at the intersection of column d and column c. Each subsequent move is listed in this format. The resolution (RE) of the game is shown in the text `RE[B+Resign]`, which means that Black resigned the game.

The AlphaGo designers amassed a massive dataset of thirty million SGF game files. The dataset wasn't randomly generated; those thirty million games were actual games played by actual people (and some computers). Whenever amateurs or professionals played Go on one of many online sites, that data was saved. It's not hard to create a Go video game; many versions of instructions and free code are posted online. All video games *can* save game data, of course. Some do; some don't. Some save your game data and use it for creating reports for the game company. The people who ran various online Go sites decided to publish their saved game data online in huge batches. Eventually, these batches were pooled, resulting in the thirty million games collected by the AlphaGo team.

The programmers used the thirty million games to "train" the model that they named *AlphaGo*. What you must remember is that people who play Go professionally spend *ages* playing Computer Go. It's how they train. Therefore, the thirty million games recorded included data from the world's greatest Go players. Millions of hours of human labor went into creating the training data—yet most versions of the AlphaGo story focus on the magic of the algorithms, not the humans who invisibly and over the course of years worked (without compensation) to create the training data.

The developers programmed AlphaGo to use a method called *Monte Carlo search* to pick a set of moves from the thirty million games that would most likely lead to a win. Then, they instructed it to use an algorithm to select the next move from the set. They also instructed it to use a different algorithm that calculated the probability of a win for each possible move in the set. The calculations happened on a scale that the human mind can barely imagine. There are $10^{170}$ possible board configurations in Go. By layering a variety of computational methods and always choosing the move with the greatest probability of success, the designers created a program that defeated the world's greatest Go players.

Is AlphaGo smart? Its designers certainly are. They solved a math problem that was so hard that it took decades of great minds to work on it. One of the amazing things about math is that it allows you to see underlying patterns in how the world works. Many, many things operate according to mathematical patterns: crystals grow in regular patterns, and cicadas hibernate underground for years and emerge when soil temperature conditions are just right, to name just two. AlphaGo is a remarkable mathematical achievement that was made possible by equally remarkable advances in computing hardware and software. AlphaGo's team of designers deserves praise for this outstanding technical achievement.

AlphaGo is not an intelligent machine, however. It has no consciousness. It does only one thing: plays a computer game. It contains data from thirty million games played by amateurs and by the world's most talented players. On some level, AlphaGo is supremely dumb. It uses brute force and the combined effort of many, many humans to defeat a single Go master. The program and its underlying computational methods will likely be deployed for other useful tasks involving massive number-crunching, and that's good for the world—but not everything in the world is a calculation.

Once we get past the mathematical and physical reality of a program like AlphaGo, we're in the realm of philosophy and future speculation. Those are very different intellectual landscapes. There are futurists who *want* AlphaGo to signify the beginning of an era in which people and machines become fused. Wanting something doesn't make it true, however.

Philosophically, there are lots of interesting questions to discuss centering on the difference between calculation and consciousness. Most people are familiar with the Turing test. Despite what the name suggests, the Turing test is not a quiz that a computer can pass to be considered intelligent. In his paper, Turing proposed a thought experiment about talking to a machine. He rejected the question "Can machines think?" as absurd and claimed it was best answered by an opinion poll. (Turing was a bit of a snob about math. Like many mathematicians then and a smaller number now, he believed in the superiority of mathematics to other intellectual pursuits.) Instead, Turing proposed an "imitation game" played by a man (A), a woman (B), and an interrogator (C). C sits in a room alone and submits typewritten queries to A and B. Turing writes: "The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either 'X is A and Y is B' or 'X is B and Y is A.'"[2]

Turing then breaks down the kind of kind of questions the interrogator is allowed to ask. One is about hair length. A, the man, wants the interrogator to make the wrong assumptions and is willing to lie. B, the woman, wants to help the interrogator and can tell him or her that she is the woman—but A can lie and say that as well. Their answers are written down, so that the quality and tone of voice cannot provide clues. Turing writes: "We now ask the question, 'What will happen when a machine takes the part of A in this game?' Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? These questions replace our original, 'Can machines think?'"

If the questioner can't tell the difference between a response provided by a human or the response provided by a machine, the computer is said to be *thinking*. For many years, this was considered foundational in computing. A vast amount of ink has been spilled trying to respond to Turing's ideas in this paper and to make a machine that can perform to Turing's specifications. However, undergirding the entire thought experiment is a philosophical and cultural misnomer that throws the entirety into question, and that

is gender. Turing's specifications do not conform to what we now understand about gender. Gender is not a binary, but a continuum. Hair length is no longer a signifier of male or female identity; anyone can rock a short haircut. Moreover, as Turing writes, "The object of the game for the third player (B) is to help the interrogator." A game to determine "intelligence," in which the woman is assigned to be the helper? And the man is told that he can lie? The underpinnings are absurd, from a critical perspective, in that both the man and woman are given gender-coded physical and moral attributes.

The philosophical underpinnings of Turing's argument are unsound. One of the most compelling counterarguments was addressed by the philosopher John Searle in an argument known as the Chinese Room. Searle summarized it in a 1989 piece in the *New York Review of Books*:

> A digital computer is a device which manipulates symbols, without any reference to their meaning or interpretation. Human beings, on the other hand, when they think, do something much more than that. A human mind has meaningful thoughts, feelings, and mental contents generally. Formal symbols by themselves can never be enough for mental contents, because the symbols, by definition, have no meaning (or interpretation, or semantics) except insofar as someone outside the system gives it to them.
>
> You can see this point by imagining a monolingual English speaker who is locked in a room with a rule book for manipulating Chinese symbols according to computer rules. In principle he can pass the Turing test for understanding Chinese, because he can produce correct Chinese symbols in response to Chinese questions. But he does not understand a word of Chinese, because he does not know what any of the symbols mean. But if he does not understand Chinese solely by virtue of running the computer program for "understanding" Chinese, then neither does any other digital computer because no computer just by running the program has anything the man does not have.[3]

Searle's argument that symbolic manipulation is not equivalent to understanding can be seen in the popularity of voice interfaces in 2017. "Conversational" interfaces are popular, but they are far from intelligent.

Amazon's Alexa and other voice-response interfaces don't understand language. They simply launch computerized sequences in response to sonic sequences, which humans call verbal commands. "Alexa, play 'California Girls'" is a voice command that a computer can follow. *Alexa* is the trigger word that tells the computer that a command is coming. *Play* is a trigger word that means "retrieve an MP3 from memory and send the command

*play* to a previously specified audio player, along with the MP3 file name." The interface is also programmed to capture whatever word comes after *play* and before the pause (the end of the command). That value is put into a variable such as *songname*, which is retrieved from memory and fed to the audio player. This process is procedural and unthreatening and shouldn't make anyone think that the machines are going to rise up and take over the world. Right now, a computer can't reliably distinguish whether it should respond to the previous command by playing Katy Perry's "California Gurls" or the Beach Boys' "California Girls." In fact, this exact problem is solved by running a popularity contest. Whichever song is played more often by all Alexa users is assumed to be the default choice. This is good for Katy Perry fans, but not so good for Beach Boys fans.

I'm going to ask you to keep the two competing ideas about narrow and general AI, and the idea of limitations, in your mind as you read. In this book, we'll stay squarely in the realm of reality: the world in which we have unintelligent computing machines that we *call* intelligent machines. However, we'll also look at how imagination—which is powerful and wonderful and exciting—sometimes confuses the way we talk about computers, data, and technology. I'd also ask you not to be disappointed like the art student at the science fair when you come up against what a colleague calls the *ghost-in-the-machine fallacy*—the reality that there is no little person or simulated brain inside the computer. There are different ways to react to this news: you can be sad that the thing you dreamed of is not possible—or you can be excited and embrace what *is* possible when artificial devices (computers) work in sync with truly intelligent beings (humans). I prefer the latter approach.