10 Artificial Intelligence

David: Martin is Mommy and Henry's real son. After I find the Blue Fairy then I can go home. Mommy will love a real boy. The Blue Fairy will make me into one.
Gigolo Joe: Is Blue Fairy Mecha, Orga, man or woman?
David: Woman.
Gigolo Joe: Woman? I know women! They sometimes ask for me by name. I know all about women. About as much as there is to know. No two are ever alike, And after they've met me, no two are ever the same. And I know where most of them can be found.
David: Where?
Gigolo Joe: Rouge City. Across the Delaware.

-: Dialog between two Artificial Intelligence entities: Gigolo Joe (played by Jude Law) and David (played by Haley Joel Osment) in the movie, Artificial Intelligence (2001), Directed by Steven Speilberg, Warner Bros.

The Question of Intelligence

The quest for the understanding of intelligence probably forms the oldest and yet to be fully understood human inquiry. With the advent of computers and robots the question of whether robots and computers can be as intelligent as humans has driven the scientific pursuits in the field of *Artificial Intelligence (AI)*. Whether a computer can be intelligent was lucidly discusses by Professor Alan Turing in 1950. To illustrate the issues underlying machine intelligence, Turing devised a thought experiment in the form of an *imitation game*. It is played with three people, a man, a woman, and an interrogator. They are all in separate rooms and interact with each other by typing text into a computer (much like the way people interact with each other over IM or other instant messaging services). The interrogator's task is to identify which person is a man (or woman). To make the game interesting, either player can try and be deceptive in giving their answers. Turing argues that a computer should be considered intelligent if it could be made to play the role of either player in the game without giving itself away. This *test* of intelligence has come to be called the *Turing Test* and has generated much activity in the community of AI researchers (see exercises below). The dialog shown above, from the movie *Artificial Intelligence*, depicts an aspect of the test of intelligence designed by Alan Turing. Based on the exchange between Gigolo Joe and David, can you conclude that they are both intelligent? Human?

After over five decades of AI research, the field has matured, and evolved in many ways. For one, the focus on intelligence is no longer limited to humans: insects and other forms of animals depict varying degrees and kinds of intelligence have been the subject of study within AI. There has also been a fruitful exchange of ideas and models between AI scientists, biologists, psychologists, cognitive scientists, neuroscientists, linguists and philosophers. You saw examples of such an influence in the models of Braitenberg vehicles introduced earlier. Given the diversity of researchers involved in AI there has also been an evolution of what AI itself is really about. We will return to this later in the chapter. First, we will give you a few examples of models that could be considered *intelligent* that are commonly used by many AI scientists.

Language Understanding

One aspect of intelligence acknowledged by many people is the use of language. People communicate with each other using a language. There are many (several thousand) languages in use on this planet. Such languages are called *natural languages*. Many interesting theories have been put forward about the origins of language itself. An interesting question to consider is: Can people communicate with computers using human (natural) languages? In other words, can a computer be made to understand language? Think about that for a minute and see if you can come up with a possible answer.

To make the question of language understanding more concrete, think of your Scribbler robot. So far, you have controlled the behavior of the robot by writing Python programs for it. Is it possible to make the Scribbler understand English so that you could interact with it in it? What would an interaction with Scribbler look like? Obviously, you would not expect to have a conversation with the Scribbler about the dinner you ate last night. However, it would probably make sense to ask it to move in a certain way? Or to ask whether it is seeing an obstacle ahead?

Do this: Write down a series of short 1-word commands like: forward, right, left, stop, etc. Create a vocabulary of commands and then write a program that inputs a command at a time interprets it and makes the Scribbler carry it out. For example:

```
You: forward
Scribbler: starts moving forward…
You: right
Scribbler starts turning right…
You: stop
…
```

Experiment with the behavior of the robot based on these commands and think about the proper interpretation that may make its behavior more natural.

You will find yourself making several assumptions about interpretation of even the simplest commands in the exercise above. For example, what happens when after you command the Scribbler to move forward, you ask it to turn right? Should the Scribbler stop going forward or should it stop and then start turning?

Decisions like these also give deep insights into our own abilities of understanding language. You can also see that, as in the case of visual perception, processing of language (or text) begins at a very primitive level: words. If the input is speech, the basic units are electrical signals, perhaps coming from a microphone. Just like processing individual pixels to try and understand the contents of an image, one has to start at a low level of representation for beginning to understand language.

Researchers working in the field of *computational linguistics* (or *natural language understanding*) have proposed many theories of language processing that can form the basis of a computational model for a Scribbler to understand a small subset of the English language. In this section, we will examine one such model which is based on the processing of syntax and semantics of language interaction. Imagine, interacting with the Scribbler using the following set of sentences:

```
You: do you see a wall?
Scribbler: No

You: Beep whenever you see a wall.
You: Turn right whenever you see a wall to your left.
You: Turn left whenever you see a wall to your right.
You: Move for 60 seconds.

[The Scribbler robot moves around for 60 seconds turning
whenever it sees a wall. It also beeps whenever it sees a
wall.]
```

Earlier, you have written Python programs that perform similar behaviors. However, now imagine interacting with the robot in the fashion described. From a physical perspective, imagine that you are sitting in front of a

computer, and you have a Bluetooth connection to the robot. The first question then becomes: Are you actually speaking or typing the above commands? From an AI perspective, both modalities are possible: You could be sitting in front of the computer and speaking into a microphone; or you could be typing those commands on the keyboard. In the first instance, you would need a speech understanding capability. Today, you can obtain software (commercial as well as freeware) that will enable you to do this. Some of these systems are capable of distinguishing accents, intonations, male or female voices etc. Indeed, speech and spoken language understanding is a fascinating field of study that combines knowledge from linguistics, signal processing, phonology, etc.

You can imagine that the end result of speaking into a computer is a piece of text that transcribes what you said. So, the question posed to the Scribbler above: *Do you see a wall?* will have to be processed and then transcribed into text. Once you have the text, that is, a string `"Do you see a wall?"` it can be further processed or analyzed to understand the *meaning* or the content of the text. The field of *computational linguistics* provides many ways of syntactic parsing, analyzing, and extracting meaning from texts. Researchers in AI itself have developed ways of representing knowledge in a computer using symbolic notations (e.g. *formal logic*). In the end, the analysis of the text will result in a `getIR()` or `getObstacle()` command to the Scribbler robot and will produce in a response shown above.

Our goal of bringing up the above scenario here is to illustrate to you various dimensions of AI research that can involve people from many different disciplines. These days, it is entirely possible even for you to design and build computer programs or systems that are capable of interacting with robots using language.

Game Playing

In the early history of AI, scientists posed several challenging tasks which if performed by computers could be used as a way of demonstrating the feasibility of machine intelligence. It was common practice to think of games

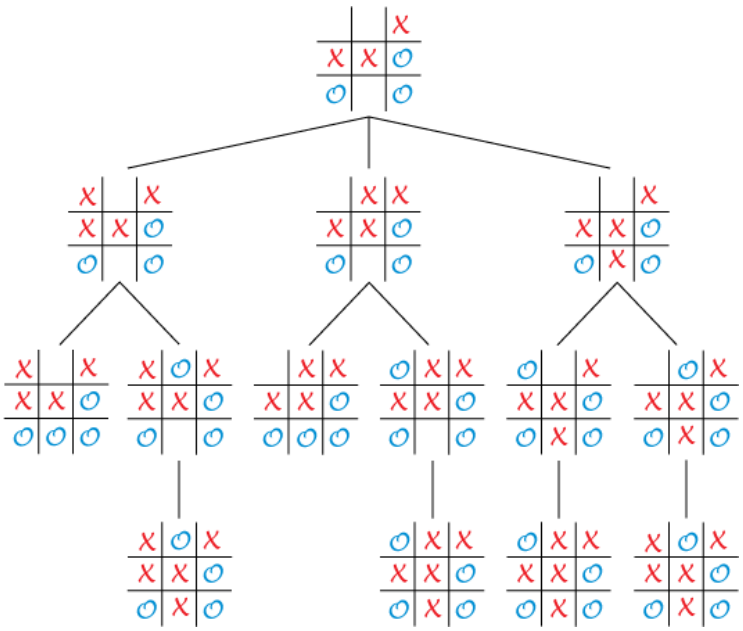
in this realm. For example, if a computer could play a game, like chess, or checkers, at the same level or better that humans we would we convinced into thinking that it was indeed feasible to think of a computer as a possible candidate for machine intelligence. Some of the earliest demonstrations of AI research included attempts at computer models for playing various games. Checkers and chess seemed to be the most popular choices, but researchers have indulged themselves into examining computer models of many popular games: poker, bridge, scrabble, backgammon, etc.

In many games, it is now possible for computer models to play at the highest levels of human performance. In Chess, for example, even though the earliest programs handily beat novices in the 1960's, it wasn't until 1996 when an IBM computer chess program, named Deep Blue, beat the world champion Gary Kasparov at a tournament-level game, though Kasparov did manage to win the match 4-2. A year later, in New York, Deep Blue beat Kasparov in a 6 game match representing the very first time a computer beat the best human player in a classical style game of chess. While these accomplishments are worthy of praise it also now clear that the quest for machine intelligence is not necessarily answered by computer game playing. This has resulted in much progress in game playing systems and game playing technology which now stands in its own right as a multi-billion dollar industry.

It turns out that in many chess-like games the general algorithm for a computer to play the game is very similar. Such games are classified as two-person zero-sum games: two people/computers play against each other and the result of the game is either a win for one player and loss for the other, or it is a draw (which makes it a zero-sum end result). In many such games, the basic strategy for making the next move is simple: look at all the possible moves I have and for each of them all the possible moves the other player might have and so on until the very end. Then, trace back from wins (or draws) and make the next move based on those desirable outcomes. You can see this easily in a simple Tic-Tac-Toe game (see picture below):

When you play against an opponent, you are anticipating possible moves down the road and then playing your own moves with those in mind. Good players are able to mentally picture the game several moves ahead. In many games, like Chess, certain recognizable situations lead to well determined outcomes and so a great part of playing a successful game also relies on the ability to recognize those situations. Looking ahead several moves in a

A Tic Tac Toe Game Tree to look for possible next moves for X



systematic manner is something computers are quite capable of doing and hence anyone (even you!) can turn them into fairly good players. The challenge lies in the number of moves you can look ahead and in the limited capacity, if time to make the next move is limited, how to choose among the best available options? These decisions lend interesting character to computer game programs and continue to be a constant source of fascination for many people. For example, a computer program to play Tic-Tac-Toe can easily look at all the possible moves all the way to the end of game in determining its next move (which, in most situations leads to a draw, given the simplicity of the

game). However, if you consider a typical game of Chess, in which each player makes an average of 32 moves and the number of feasible moves available at any time averages around 10, you would soon realize that the computer would have to examine something of the order of 10^65 board positions before making a move! This, even for the fastest computers available today, will take several gazillion years! More on that later. But, to play an interesting two-person zero-sum game, it is not essential to look so far ahead.

In Chapter 7, you saw an example of a program that played the game of Paper-Scissors-Rock against a human user. In that version, the program's choice strategy for picking an object was completely random. We reproduce that section of the program here:

```
…
items = ["Paper", "Scissors", "Rock"]
…
# Computer makes a selection
myChoice = items[randint(0, 2)]
…
```

In the above program segment, `myChoice` is the program's choice. As you can see, the program uses a random number to select its object. That is, the likelihood of picking any of the three objects is 0.33 or 33%. The game and winning strategies for this game have been extensively studied. Some strategies rely on detecting patterns in human choice behavior. Even though we may not realize it there are patterns in our seemingly random behavior. Computer programs can easily track such behavior patterns by keeping long histories of player's choices, detect them, and then design strategies to beat those patterns. This has been shown to work quite effectively. It involves recording player's choices and searching through them (see Exercises). Another strategy is to study human choice statistics in this game. Before we present you with some data, do the exercise suggested below:

Do This: Play the game against a few people, Play several dozen rounds. Record the choices made by each player (just write a P/S/R in two columns). Once done, compute the percentages of each object picked. Now read on.

It turns out that most casual human players are more prone towards picking Rock than Paper or Scissors. In fact, various analyses suggest that 36% of the time people tend to pick Rock, 30% Paper, and 34% Scissors. This suggests that RPS is not merely a game of chance there is room for some strategies at winning. Believe it or not, there are world championships of PSR held each year. Even a simple game like this has numerous possibilities. We can use some of this information, for instance, to make our program smarter or better adept at playing the game. All we have to do is instead of using a fair 33% chance of selecting each object we can skew the chances of selection based on people's preferences. Thus, if 36% of the time people tend to pick Rock, it would be better for our program to pick Paper 36% of the time since Paper beats Rock. Similarly, our program should pick Scissors 30% of the time to match the chance of beating Paper, and pick Rock 34% of the time to match the chances of beating Paper. We can bias the random number generator using these percentages as follows:

First generate a random number in the range 0..99
If the number generated is in the range 0..29, select Scissors (30%)
If the number generated is in the range 30..63, select Rock (34%)
If the number generated is in the range 64..99, select Paper (36%)

The above strategy of biasing the random selection can be implemented as follows:

```
def mySelection():

    # First generate a random number in the range 0..99
    n = randrange(0, 100)

    # If the n is in range 0..29, select Scissors
    if n <= 29:
        return "Scissors"
    elif n <= 63:
```

```
        # if n in range 30..63, select Rock
        return "Rock"
    else:
        return "Paper"
```

Do This: Modify your RPS program from Chapter 7 to use this strategy. Play the game several times. Does it perform much better that the previous version? You will have to test this by collecting data from both versions against several people (make sure they are novices!).

Another strategy that people use is based upon the following observation:

After many rounds, people tend to make the move that would have beaten their own previous move.

That is, if say a player picks Paper. Their next pick will be Scissors. A computer program or a player playing against this player should then pick Rock to beat Scissors. Since the relationship between the choices is cyclical the strategy can be implemented by picking the thing that beats the opponent's pervious move beats. Paper beats Rock. Therefore since the player's previous move was Paper, your program can pick Rock in anticipation of the player's pick of Scissors. Try to think over this carefully and make sure your head is not spinning by the end of it. If a player can spot this they can use this as a winning strategy. We will leave the implementation the of last strategy as an exercise. The exercises also suggest another strategy.

The point of the above examples is that using strategies in your programs you can make your programs smarter or more intelligent. Deliberately, we have started to use the term intelligence a little more loosely than what Alan Turing implied in his famous essay. Many people would argue that these programs are not intelligent in the ultimate sense of the word. We agree. However, writing smarter programs is a natural activity. If the programs incorporate strategies or heuristics that people would use when they are doing the same activity, then the programs have some form of artificial intelligence in them. Even if the strategy used by the program is nothing like what people would use, but it would make the program smarter or better, we would call it

artificial intelligence. Many people would disagree with this latter claim. To some, the quest for figuring out intelligence is limited to the understanding of intelligence in humans (and other animals). In AI both points of view are quite prevalent and make for some passionate debates among scholars.

Learning

Here we will give you an overview of machine learning, introduce you to the idea of computational neural networks, and then show you how using the Myro modules for neural networks, you can design a learning program for your Scribbler robot to learn how to avoid obstacles. Yet to be written…

Discussion

The very idea of considering a computer as an intelligent device has its foundations in the general purpose nature of computers. By changing the program the same computer can be made to behave in many different ways. At the core of it a computer is just a symbol manipulator: manipulating encodings for numbers, or letters, or images, etc. It is postulated that the human brain is also a symbol manipulator. The foundations of AI lie in the fact that most intelligent systems are physical symbol systems and since a computer is a general purpose symbol manipulator, it can be used for studying or simulating intelligence.

Myro Review

There were no new Myro features introduced in this chapter. Actually, when the chapter is complete it will have Myro primitives for neural nets/connx described here.

Python review

No new Python features were introduced in this chapter.

Exercises

1. Read Alan Turing's paper Computing Machinery and Intelligence. You can easily find a copy of it by searching on the web.

2. More to come…