PART OF THE PICTURE: Artificial Intelligence

BY KEITH VANDER LINDEN, CALVIN COLLEGE

Just over an hour into the sixth game of their chess match, Garry Kasparov, the reigning world champion, conceded defeat to Deep Blue, a chess-playing computer developed by IBM corporation. Kasparov lost the match, held in May of 1997, 3.5 games to 2.5 games. It marked the first time a computer program had defeated a world chess champion in anything approaching tournament conditions.

Although this result came as a surprise to many, it has been clear for some time that a computer would eventually beat a world champion player. Since their introduction to tournament play in the late 1960's, chess programs have made steady progress, defeating a chess master in 1983, a grandmaster in 1988, and now the world champion. This is by no means the end of the story, however. There are reservations concerning the validity of this most recent match: it was not really a tournament setting with multiple players, Kasparov was not allowed to study Deep Blue's previous matches, and he was under considerable pressure to hold off a perceived "attack on humanity" by computer programs. Nevertheless, another milestone has been passed.

The construction of game-playing programs such as Deep Blue is part of a subfield of Computer Science known as **Artificial Intelligence** or **AI.** Roughly speaking, AI is an attempt to program computers to perform intelligent tasks. Giving a precise definition of AI is difficult, however, and most AI textbooks spend a laborious opening chapter attempting to characterize the field. This difficulty comes about for two reasons: (1) because intelligent behavior is complex and hard to define; and (2) because the styles of programming used to implement this behavior are diverse.

INTELLIGENCE

We all have a general notion of what intelligence is. We presume to know who has it and who doesn't. Garry Kasparov, for example, is intelligent by any measure. He is the world champion of chess, a game long seen as a pinnacle of human intellectual achievement. But what about Deep Blue? Is it intelligent? It beat Garry Kasparov in a chess match; this must mean something.

Clearly, Deep Blue has performed an intelligent feat. It has done so, however, in a very narrow domain. This is one thing that distinguishes its accomplishment from general human intelligence. Intelligent behavior, the "I" in AI, is diverse. Humans display profoundly complex behavior in many areas, all of which have been topics of study in AI, including:

- *Reasoning and problem solving*—Humans are able to reason about their world and to plan their own actions accordingly. This encompasses a variety of activities, including game playing, mathematical theorem proving, and the planning of actions. Examples of systems which perform these tasks include Deep Blue, the game-playing system just discussed; EQP, a system which, in 1996, proved a theorem

that no human has successfully proven; and SPIKE, a system which plans observation schedules for the Hubble space telescope.

- *Memory*—Humans are able to remember things about their world. In AI, the study of this area is called **knowledge representation.** It serves as the foundation of all reasoning and problem solving. Deep Blue, for example, represents considerable knowledge about chess but, it must be said, little or nothing about anything else.
- *Motion and manipulation*—Humans are able to perform actions in their world. The area of AI which deals with this is called **robotics.** This area, much popularized in science fiction, includes the development of robotic arms for assembly lines, or of fully mobile agents such as Robosoft's Auto VacC, an autonomous vacuum cleaner.
- *Perception*—Humans are able to see and to hear. The areas of AI concerned with these behaviors are called **computer vision** and **speech recognition.** They have been successful in fielding systems, which convert written characters into ASCII characters, called optical character recognition systems and systems which convert spoken words into ASCII words, called speech-to-text dictation systems.
- *Language processing*—Humans are able to process natural, human languages. In AI, the behaviors studied include the ability to understand language **(natural language understanding),** the ability to produce language **(natural language generation),** and the ability to translate from one language to another **(machine translation).** Examples of this include M_t_o, a system which generates weather reports in English and French, and Systran, a machine translation system used by the European Commission.
- *Learning*—Humans are also able to learn from past experiences. In AI, the study of this phenomenon is called **machine learning.** One practical application of this involves **data mining.** Data mining tools attempt to find consistent patterns in large amounts of data. One such tool, ISL's Clementine, has been trained to predict the audience-share for prospective new television shows on the BBC.

Deep Blue can play chess well, but doesn't exhibit any of the other behaviors just given. It can't even move its own chess pieces. At first glance, however, many of these other areas don't appear to be all that difficult, particularly when compared with chess. My four-year old son, for example, can't play chess very well yet, but he does exhibit all the other behaviors with relative ease. He remembers things, he perceives and manipulates things, he processes language, and he learns. These have all come naturally to him. Surely with a little extra work, Deep Blue could do them as well. This is a critical misconception. Just because a task is easy for people doesn't necessarily means that it is *simple*. In fact, some of the things humans find extremely easy to perform have turned out to be among the hardest to program. For example, in the area of natural language processing, we have not succeeded in producing a system capable of engaging in meaningful conversation except in narrowly defined contexts. Conversely, some of the greatest successes in AI have been in areas seen as requiring great skill such as chess playing, calculus, and medical diagnosis.

This paradox of sorts is part of the reason that early researchers in AI grossly underestimated the difficulties in AI programming. Indeed, the field of AI has been greatly damaged by overly-ambitious expectations. In the mid-to-late 50's, for example, many people felt that machine translation programs were just around the corner. In retrospect, this was an extraordinary claim given that at the time we barely knew how to translate FORTRAN into assembly language.

PROGRAMMING TECHNIQUES

Another point that distinguishes Deep Blue's accomplishment from general human intelligence is the mechanism by which it operates, the "A" in its AI. As there is a wide range of intelligent behavior, there is also a wide range of programming techniques used to implement them "artificially." These techniques include:

- *Heuristic search*—AI programs are frequently designed to consider a number of choices. This is called **search** because the program is said to search through a space of possible choices and their consequences. For example, before making a choice, Deep Blue considers many moves and what their consequences might be many steps into the future. It uses a 256-processor architecture to do this, considering on the order of 200 million board positions per second. Search alone is seldom sufficient, however. The nature of chess, for example, dictates that Deep Blue be selective in what moves it considers, and how far down the road it considers them. There are simply too many alternatives to consider them all. It, therefore, tends to ignore the less promising ones, and to focus on a few options. These options are chosen based on **heuristics,** or rules of thumb, such as "first, consider moves that gain control of the center of the board."
- *Logic programming*—This approach involves representing knowledge in a well-defined format and performing logical inferences on it. For example, EQP, the theorem-proving program mentioned earlier, uses such an approach to prove theorems. It takes a set of given knowledge, and attempts to derive the theorem logically from this knowledge. The programming language **Prolog** has been specifically designed to support logic programming.
- *Expert systems*—Expert systems encode knowledge elicited from an expert in some domain. Many of these systems have become commercial successes, including XCON (a system which configures computer components for DEC equipment), and SBDS (a system which diagnoses electrical problems in Ford automobile engines).
- *Neural networks*—The techniques given so far are programmed on computers which are built with digital circuitry. The human brain, on the other hand, is very different. It is constructed of very simple brain cells, called neurons, which are highly interconnected with one another. Computer models of this structure are called **neural networks** or **connectionist systems.** They display radically different characteristics from traditional programming techniques.

These techniques are not mutually exclusive, and may be used individually or in combinations to implement the intelligent behaviors discussed.

EXAMPLE: THE NOT-ONE GAME

Early researchers in AI frequently used games such as chess as a vehicle for their research. Games tend to be limited in scope and to have well-defined rules. Chess, for example, has a 12 × 12 board with a finite set of pieces, where each piece has a well-defined set of possible moves. This allowed the researchers to avoid the problems involved in modeling the other areas of intelligent behavior. We, too, will take advantage of this and implement a program that plays **not-one,** a simple dice game for two players.

In not-one, each player takes a turn in which they roll two dice and remember the sum of the values as their first roll. They may roll again as many times as they would like, with the object of getting the highest roll possible. The only stipulation is that they are forced to quit with a score of 0 for the turn if they roll

the same sum as they did for their first roll. The player with the highest score after 12 turns wins.

We'll start by implementing a turn-taking driver program. This driver runs a 12-turn game, printing out the scores at the end of each turn:

POP 11-1 A Driver for Not-One.

```
int main()
{
  cout << "\nReady to play some Not-One!\n" << endl;

  Player1 player1("Fast");
  Player2 player2("Random");

  int player1Score = 0,                 // initialize
      player2Score = 0,                 //  counters
      player1LastRoll = 0,
      player2LastRoll = 0;

  for (int i = 1; i <= 12; i++)         // play 12 rounds
  {
                                        //  the 1st player's turn
     cout << "----------------------------------------\n"
          << player1.getName() << ", turn " << i;
     player1LastRoll = player1.takeTurn(player2LastRoll);
     player1Score += player1LastRoll;
                                        //   the 2nd player's turn
     cout << "----------------------------------------\n"
          << player2.getName() << ", turn " << i;
     player2LastRoll = player2.takeTurn(player1LastRoll);
     player2Score += player2LastRoll;
                                        //   summarize round
     cout << "****************************************\n"
          << " Turn " << i << " - "
          << player1.getName() << ": " << player1Score << "; "
          << player2.getName() << ": " << player2Score
          << "\n****************************************\n"
          << endl;
  }
}
```

Notice that this driver makes use of two user-defined classes, `Player1` and `Player2`, which implement the methods `takeTurn()` and `getName()`. We have created two classes rather than one so that the player that goes first may have a different strategy from the player that goes second. The `takeTurn()` method for each player is as follows:

POP 11-2 A Turn-Taking Member Function for Not-One.

```
int Player::takeTurn(int opponentsLastRoll)
{
  RandomInt die1(1, 6);          // Declare the random dice.
  RandomInt die2(1, 6);

  myCurrentRoll = 0;
  myFirstRoll = die1 + die2;
  int scoreThisTurn = myFirstRoll;
  cout << "\n First Roll: " << die1 << " + " << die2
          << " = " << die1 + die2;

  updateData(opponentsLastRoll));// Keep track of opponent if desired.

  for (;;)                       // Continue rolling until Stop()
  {                              // returns true.
     if ( stop() )
     {
        cout << ". Stopping.\n" << endl;
        break;
     }
     else
        cout << ". Continuing . . . ";

     myCurrentRoll = die1.generate() + die2.generate();
     cout << "\n Next Roll: " << die1 << " + " << die2
          << " = " << die1 + die2;

     if (myCurrentRoll == myFirstRoll)
     {
        cout << ". OOOOPS!\n" << endl;
        scoreThisTurn = 0;
        break;
     }
     else
        scoreThisTurn = max(scoreThisTurn, myCurrentRoll);
  }

  raiseScore (scoreThisTurn);
  return scoreThisTurn;
}
```

The `takeTurn()` method declares two dice of type `RandomInt` (see the Web site described in the preface) and implements the player's turn with a loop that continues until either the player decides to stop, as determined by the `stop()` method, or the player rolls the first roll again. The appropriate score is then returned.

There are many strategies which may be implemented for this game. Recall that Deep Blue used heu-

ristic search in its strategy. That was appropriate in chess, but it is not as useful here because not-one decisions are, for the most part, made independently of what the other player is likely to do in the future. We will, therefore, use a scaled-down expert system approach in which we encode knowledge from expert players. The simplest strategy is to encode the somewhat dubious "knowledge" that a player should always accept his or her first roll. This strategy has the virtue of simplicity, and also the advantage that it never gets a 0 for a turn because it never risks re-rolling. It is implemented with the following `stop()` method for `Player1`:

```
bool Player1::stop() const
{
  return true;
}
```

Another simple strategy is to randomly determine whether to go on or not. This is implemented in the following `stop()` method for `Player2`:

```
bool Player2::stop() const
{
  RandomInt choice(0,1);    // Declare a random coin.

  if (choice)
    return true;
  else
    return false;
}
```

A partial output for a game between these two strategies is shown here. This output shows the first three turns and uses the names "Fast" and "Random" for the two strategies.

POP 11-3 A Sample Output of a Not-One Game.

```
Ready to play some Not-One!

----------------------------------------
Fast, turn 1
 First Roll: 2 + 2 = 4.  Stopping.
----------------------------------------
Random, turn 1
 First Roll: 2 + 4 = 6.  Continuing . . .
 Next Roll: 4 + 4 = 8.  Continuing . . .
 Next Roll: 4 + 4 = 8.  Continuing . . .
 Next Roll: 3 + 3 = 6.  OOOOPS!

****************************************
 Turn 1 - Fast: 4; random: 0
****************************************
```

```
----------------------------------------
Fast, turn 2
 First Roll: 1 + 1 = 2.  Stopping.

----------------------------------------
Random, turn 2
 First Roll: 4 + 4 = 8.  Stopping.

****************************************
 Turn 2 = Fast: 6; random: 8
****************************************

----------------------------------------
Fast, turn 3
 First Roll: 4 + 4 = 8.  Stopping.

----------------------------------------
Random, turn 3
 First Roll: 6 + 1 = 7.  Continuing . . .
 Next Roll: 1 + 2 = 3.  Continuing . . .
 Next Roll: 1 + 2 = 3.  Stopping.

****************************************
  Turn 3 - Fast: 14; random: 15
****************************************
```

There are clearly more effective strategies for this game, which would most likely include additional expert knowledge about how to act in certain specific situations. Expert players, for example, might likely take more risks if they are far behind near the end of a game, or they might become more conservative if they have a "comfortable" lead. Other approaches might involve a statistical analysis of "optimal" choice. Implementing this additional knowledge is left as an exercise.

FURTHER READING

If you are interested in reading further on Artificial Intelligence, consider going to the following sources:

- Russell and Norvig's text *Artificial Intelligence, A Modern Approach*, Prentice Hall, 1995—This is a good, comprehensive introduction to the field, which discusses not only the computer science in AI, but also the influence of other disciplines such as philosophy, psychology, and linguistics. They also discuss many of the concepts and example systems mentioned in this section.
- Some of the early papers in AI are still as incisive today as they were when they first came out—Turing wrote a landmark paper on the nature of AI, "Computing Machinery and Intelligence," *Mind*, 59:433–460, 1950. Searle wrote an oft-cited critique of Turing's vision, "Minds, Brains and Programs," *Behavioral and Brain Sciences*, 3:417–424, 1980.
- There are also extensive materials available on the internet—Carnegie Mellon's AI repository at

```
http://www.cs.cmu.edu/Groups/AI/html/repository.html
```

has an extensive collection of documents and systems. The Kasparov vs. Deep Blue chess match is discussed at length at IBM's site,

```
http://www.chess.ibm.com/
```

EXERCISES

1. Using the not-one skeleton given in this section, program the not-one game with better heuristic functions for player 1 and player 2. Try pitting your strategy against those of your colleagues.
2. Write a strategy that allows the user to play manually against other strategies.