he
ne of the
robot
where a
gs pro-
it sensor
lations.
as

Chapter 20

Board Games

Solutions in this chapter:

- Playing Tic-Tac-Toe
- Playing Chess
- Playing Other Board Games

Introduction

Board games are among the most challenging projects you can take on with your MINDSTORMS kit. The RCX does have the power to run software that plays Tic-Tac-Toe, Checkers, and even Chess (at some level), but this doesn't mean that such a program is easy to write and test.

We believe that the hardest but most important part of the job is the creation of the interface between your robot and the physical world. Though running a Chess program on the RCX is quite a challenging task in itself, having a robot physically interpret and interact with the data is another giant step.

The method you choose to represent the board and make your robot actually move the pieces determines in large part the technical difficulties you will have to solve. In this chapter, we are going to describe some of the possible approaches, from the easiest to the trickiest.

Playing Tic-Tac-Toe

As we described in our introduction to the book, in October 1999 we attended the first Mindfest gathering at the Massachusetts Institute of Technology (MIT) at the Media Lab facility.

The Mindfest event featured many activities: lectures, workshops, a construction zone... There was also a large exhibition area were the participants could show off their MINDSTORMS creations.

A couple of months before the event, we had already booked the plane and the hotel, and asked for a table in the exhibition area, but hadn't yet prepared our robot. Showing brilliant intuition, Marco Beri, the third member of our small group, came up with the idea of building a Tic-Tac-Toe machine, a robot able to play a board version of the well-known game. We immediately felt it was the right idea: Board games have been historically considered a good test for machine intelligence, so even if Tic-Tac-Toe can't compare to Chess in complexity we thought it was the right project to present in the "temple" of AI at MIT.

Marco wrote the software, we built the hardware, and just a few days before leaving, we met and refined our prodigy. Our robot, named TTT, worked perfectly and aroused much interest.

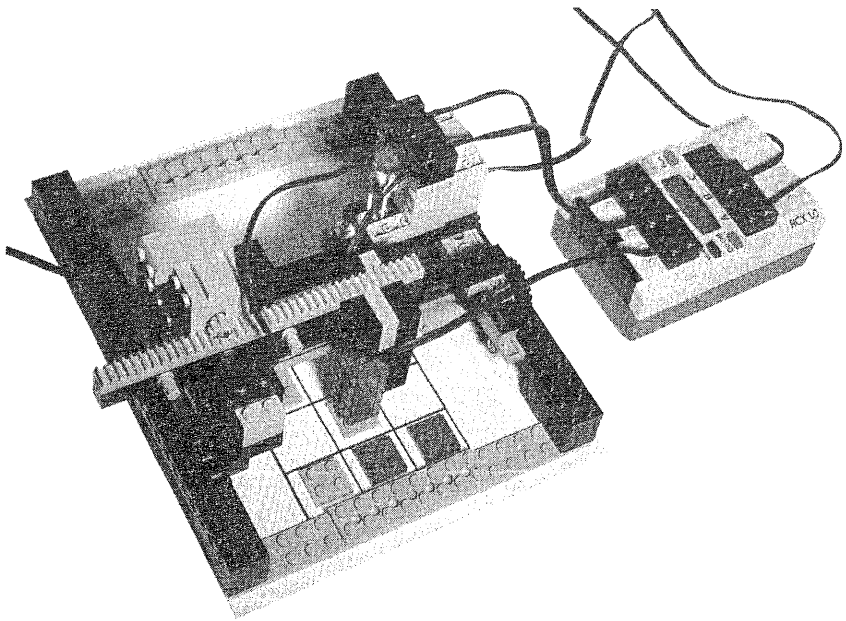
Though we made an effort to keep the requirements minimal, our machine used many extra LEGO parts. For this reason, we decided to build a new, simplified version for use in this book. The description of our original version is still online, however; you can find the link to it in Appendix A.

Building the Hardware

To keep the project replicable using only parts from the MINDSTORMS kit, we gave up the idea of making our robot physically mark its moves with a pen or with pieces. Thus this robot just indicates its move and requires your assistance in recording it on the paper.

If you look at Figure 20.1, you will probably recognize the structure: It's the Maze Solver of Chapter 19, with just some minor modifications. (We won't describe the whole robot again, just the changes we made, referring you to Chapter 19 for the remaining details.)

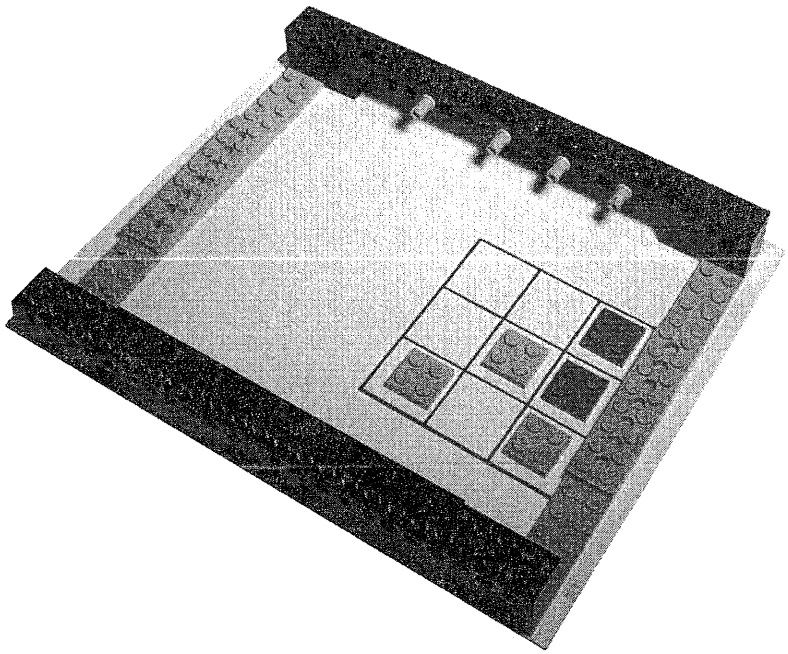
Figure 20.1 The Tic-Tac-Toe Player



The main difference concerns the number of pegs used to mark the position along the X and Y axis. In this project, you need just three stops in each direction, and a fourth for the resting position of the robot. We also removed the baseplate and linked the two longitudinal rails with some plates.

The board is represented by a white sheet of paper, the ideal surface when looking for the highest contrast in light readings. On the sheet, we traced the Tic-Tac-Toe scheme. The centers of its squares align with the stop pegs placed along the rails (Figure 20.2).

Figure 20.2 The Tic-Tac-Toe Board



This robot is designed to read the light value in the center of the squares. You can either mark the moves placing thin LEGO pieces to represent Xs and Os, or drawing large dots with colored markers. In both cases, be sure that the light sensor is very close to the surface, otherwise its readings will be badly affected by the ambient light.

We tested our robot using the 2 x 2 plates included in MINDSTORMS as markers: The gray ones for one team and the blue ones for the other. (You can use green plates with the blue, since they read as almost the same value under the light sensor.)

Writing the Program

Our robot need not know that a game is made of many moves. Every run of the program is a single move, made of the following steps:

1. The robot scans the board, storing a copy of it in a memory array.
2. It evaluates the situation and decides what move to make.
3. It performs the move.

When the user presses the Run button, the robot moves from its resting position and goes to the first row and first column of the board (second peg on the Y axis, first peg on the X axis). It stops there and reads the value of the light sensor. We suggest you use raw values, because, as explained in Chapter 4, they provide finer granularity. In our version, there's a clear-cut division among the readings of the white sheet, the gray plate, and the blue plate, definitely enough to distinguish the three cases with no ambiguity. It's crucial you place the plates directly below the reading positions of the sensor; you can make the placement more exact by marking the squares on the sheet with a thin black line.

Having scanned and stored the first square, the robot will move to the second one, then to the third and so on until the whole board has been scanned. Our TTT software, written in NQC, assigned each square to a variable, but it's possible to use a much more compact representation using individual bits, as demonstrated by Antonio Ianiero's YATTT (Yet Another Tic-Tac-Toe) that employs only two variables for the entire board (see Appendix A).

As for the strategy, when properly played, Tic-Tac-Toe ends in a tied game in which nobody wins. The following list enumerates, in order of priority, the steps to play a perfect defensive game:

1. Check if there's any move that makes you win, a square that completes a row, a column or a diagonal of three.
2. Block the opponent if there's any row, column, or diagonal with two of his pieces and one empty square.
3. In the case where you have a piece in the center and the opponent has two pieces at the ends of a diagonal, choose one of the four central squares of the external rows and columns to force him to block you.
4. If the central square is free, play there.
5. If one corner is free, play there.
6. Play in any free square.

Once the robot has figured out its move, it goes to that particular square and beeps to show it wants to play there, then returns to its resting position.

Practicing with the described strategy is a good idea; take a sheet of paper and play a few games against a friend, or by yourself, following the suggested steps as if you were the robot. This will make you familiar with the possible board situations and the moves required to oppose the attacks.

Don't forget to add a melody to your program that plays when the robot wins; a tune for defeat shouldn't be necessary because it never loses (unless the human player cheats!).

Improving Your Game

The hardware of the robot offers many possibilities for variations and improvements. If you have enough racks, you can cover the side rails with them and use gears in place of wheels, the latter having an unpleasant tendency to slip.

Using a third motor, you can equip your robot with pliers to carry and drop its own pieces, as in our original version. The Mindfest TTT still needed the help of a human assistant to load the piece in a special platform where the robot caught it, but it's possible to devise an automatic dispenser that contains a stack of pieces and drops one on demand.

On the software side, the strategy we described will never let your robot lose, but it's not particularly aggressive either. You can improve it in this area so your robot tries to confuse its opponent whenever possible.

If you have a solid background in programming (and in AI in particular), you can develop a *learning* version of this robot. It would start with no knowledge, playing purely at random in the beginning, but learning from its own mistakes and becoming better and better as the number of played games increases. In our opinion, this is quite a difficult but very impressive and instructive improvement, that makes the robot much more attractive to see in action.

Playing Chess

It's a big step from Tic-Tac-Toe to Chess, but people have succeeded in writing a version for the RCX, proving that the goal is definitely achievable (see Appendix A).

We didn't test the robots in this chapter with the proper Chess-playing software but instead used a reduced set of moves to check the mechanics. We did, however, make an effort to present ideas about interfacing a Chess software with the real world. As always, too, these robots offer tips and suggestions applicable to other areas.

For example, the visual interface employs the Fiber-Optic System (FOS) as an input/output device, using it to emulate a rotation sensor (see Chapter 4) and at the same time give a visual indication of the moves. This demonstrates that the unit, usually considered solely decorative, has greater potential: You can use its eight fibers anytime you want to show the user of your robot a value between 1 and 8.

The robotic arm of Broad Blue, our mechanical Chess interface, is a good example of a system which addresses the points of a plane using an alternative to the Cartesian scheme. In fact, while our Tic-Tac-Toe machine reaches the squares on the board using a combination of two linear movements, Broad Blue uses a combination of two angles. This scheme is very suitable for robotic cranes and grabbing arms designed for various applications, including those aimed at emulating a human arm, whose structure is based on the same principle.

Building a Visual Interface

Comparing Chess to Tic-Tac-Toe, the first thought that comes to mind is that the scan-the-board approach is very complex, if not impossible, to implement. How could a robot tell one piece from another? Even if, with specially coded pieces, you succeeded in the task, the time required for the robot to scan 64 squares would make the game quite boring.

This means you must use a different technique. The most obvious solution is that your robot keeps a copy of the board situation in its memory, updating it with the moves of the players. It knows its own moves, thus all it needs is an input about those of its human opponent. With this approach, the robot no longer needs to scan the board and differentiate the chess pieces, because it uses the copy of the board contained in its memory.

The standard convention to describe board situations in Chess is based on a simple coordinate system where rows are numbered 1 to 8 and columns A to H. Thus all you need is a way to choose a starting column-row pair, that points to the piece the human player wants to move, and a second column-row pair that outlines its destination.

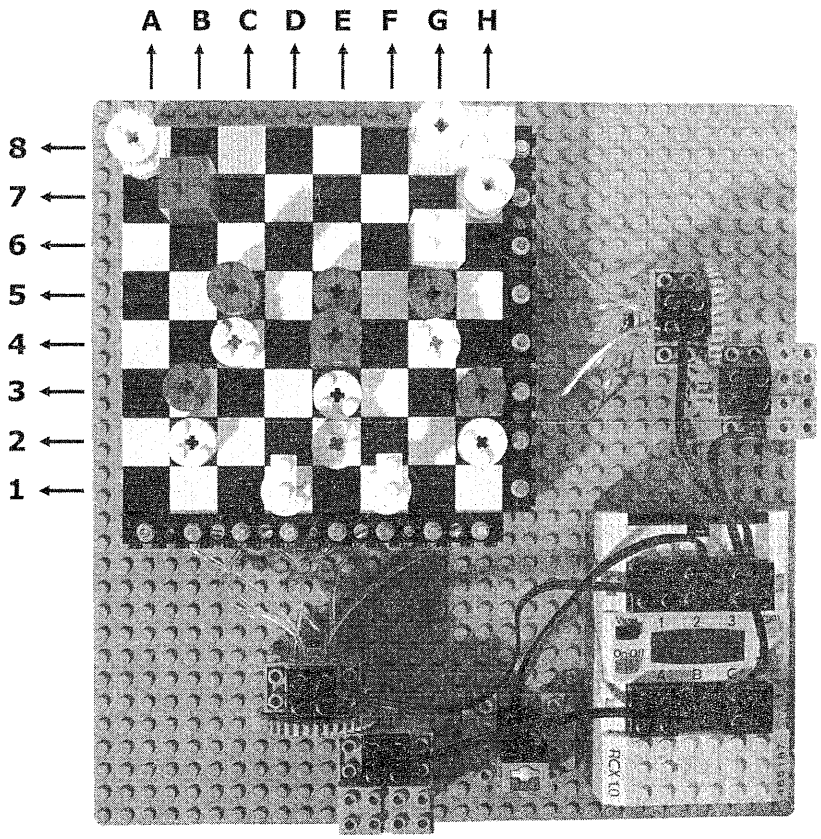
Our first Chess interface works exactly like that. We used the Fiber-Optic System (FOS) to display the selected coordinates beside a small all-LEGO board (Figure 20.3).

An FOS unit contains a rotor which has a tiny red light. Wiring the unit to a power source, like the output port of an RCX, turns the light on. The casing of the FOS device has an axle-hole in its center, through which you can rotate the inner rotor. This rotation aligns the light with one of eight possible holes, from which you can drive the beam of light into the LEGO optic fibers.

In our setup, the FOS units connect to two input ports of the RCX, which are configured as light sensors. The electrical current the units receive from the RCX, aimed at power activated sensors, is enough to make their LED turn on. At the same time, the fact that they are attached to input ports allow you to read their state. In fact, the FOS unit returns two different values, the first when the rotating

light is aligned with one of the eight holes, and the second when the light is hidden between two holes. This is the property that makes them good candidates to emulate rotation sensors: You can implement an internal counter which is incremented every time you detect a transition from one value to the other (see Chapter 4). The FOS can't tell you the direction, but if you couple it with a motor, like we did, you know in which direction you are making it turn. In our robot, each FOS unit is powered by its independent motor through a 1:5 reduction stage.

Figure 20.3 The Chess Visual Interface



Controlling the FOS from the program is quite straightforward. The first thing you have to do is display the readings returned by the FOS; you will discover they assume only two very different values, one corresponding to the eight output holes and another to the intermediate positions. Use the average of the two as the threshold of the two states. In our case, using raw values, we defined a constant THRSH equal to 775. The following NQC subroutine moves the FOS one step:

```
#define THRSH 775
int pos;

void FOS_step ()
{
  OnFwd(OUT_A);
  while(SENSOR_1 < THRSH);
  while(SENSOR_1 > THRSH);
  Off(OUT_A);
  pos++;
  if (pos > 8)
    pos=1;
}
```

Every time you call this subroutine, the light will move one step, and the inner variable `pos` will keep track of its position (provided that it started from a known point).

To input the user's moves to the RCX, we used an interface as simple as a single touch sensor. This is how the "dialog" between the players unfolds:

1. The user clicks to choose the column of the piece to move. For every click, the FOS increments one position. When okay, the user double-clicks.
2. Now the touch sensor controls the row position, and in a similar way the user selects the row of the piece to move.
3. Then he inputs the destination column and the destination row of the chosen piece.
4. The program records the required move and checks if it's valid, confirming the result with an appropriate sound. If the move is rejected, the procedure starts again from Step 1.
5. The RCX evaluates its own move, and shows the coordinates of the starting square using the FOS lights. It then waits for a confirmation click from the user.
6. Now it shows the destination square, and waits for another confirmation click.

You can speed up the input process making the program show only the valid rows and columns where the user has pieces for Steps 1 and 2, and the valid destinations for the chosen piece during Steps 3 and 4.

It's possible to improve the communication protocol, too; for example, allowing decrements of position when the sensor gets pressed for a "long" time instead of receiving a fast click.

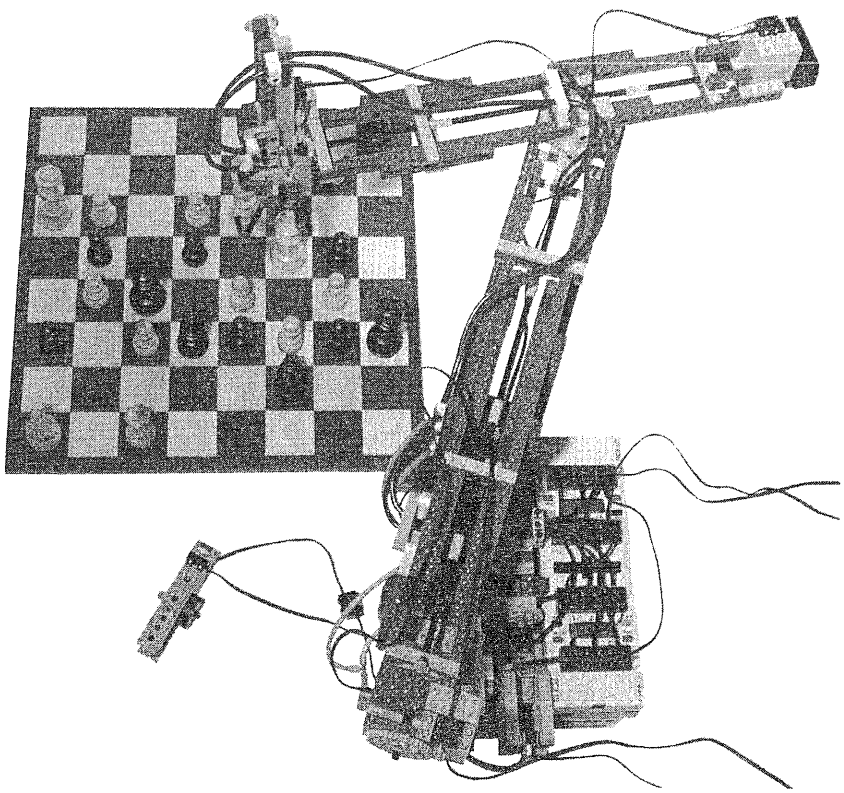
Building a Mechanical Interface

Suppose you're not satisfied by the previous simple visual interface, and want your Chess robot to really move its pieces. What difficulties can you expect?

Your system needs at least three degrees of freedom, two to get to all the squares of the board, and a third for the vertical movement necessary to lift and put down the pieces. Moreover, it needs some grabbing ability to handle the pieces.

The size of the board and the shape of the pieces greatly affect the design of your machine. We challenged ourselves by building a robot able to play on a regular wooden Chess set, and this led to an extra-large robot (Figure 20.4).

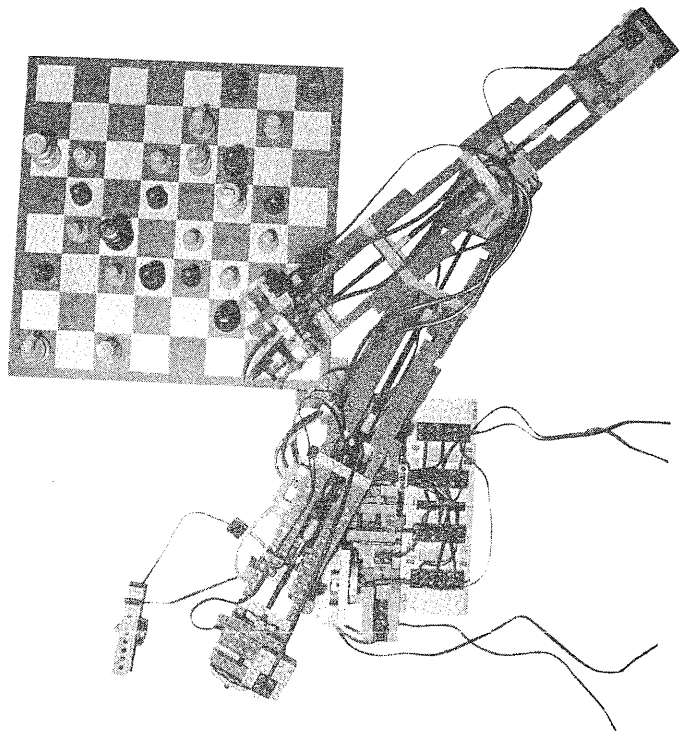
Figure 20.4 The Chess Mechanical Interface



The first thing you notice is that this time we didn't use a Cartesian system to address the square. Our robotic arm emulates the kind of movement a human arm uses, with two articulations that correspond to the shoulder and the elbow. Even the size is not too far from that of a human arm!

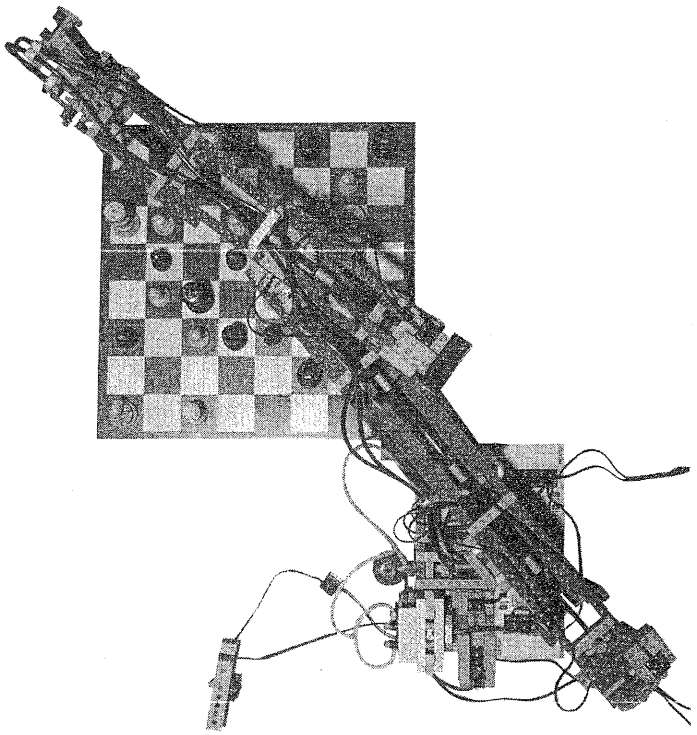
As we made wide use of blue parts, we decided to name this machine Broad Blue. (It's a play on words, considering "Deep Blue" was the name of the famous IBM computer that was the first defeat Chess world champion Garry Kasparov!) This system is able to address any point on the Chessboard, as shown in Figures 20.5 and 20.6.

Figure 20.5 Broad Blue Plays in H1



The pliers at the end of the arm must be capable of a long vertical range, since the pieces must be lifted high enough so they don't touch the other pieces on the board while the arm moves. Their jaws have to open enough to capture the pieces, but not so much that they involuntarily shift the adjacent ones. At the same time, the pliers must be able to grab all the different pieces, from the Pawn to the King.

Figure 20.6 Broad Blue Plays in A8



We used a rack and pinion assembly for the lifting mechanism, and a small pneumatic cylinder for the pliers (Figure 20.7).

On the other side of the lifting mechanism you see a touch sensor that gets closed by two cams at the extreme positions of the rack (Figure 20.8).

The motor that operates the lifting system lies at the opposite end of the secondary arm, behind the turntable (Figure 20.9), and uses a worm gear-to-16t reduction stage.

The advantage of this configuration is that the weight of the motor acts as a counterweight to the mass of the lifting mechanism. To this same purpose, we also added a weighed brick side to the motor. It's very important that the secondary arm is well-balanced, otherwise it will introduce a strong torsion (twisting) on the primary one, and torsion is much more difficult than weight to oppose and compensate for. In the same picture, note also that the secondary turntable gets turned by an 8t gear. This is connected through a pair of bevel gears to the long joined axle that reaches the motor at the opposite side of the primary arm.

Figure 20.7 The Lifting Mechanism

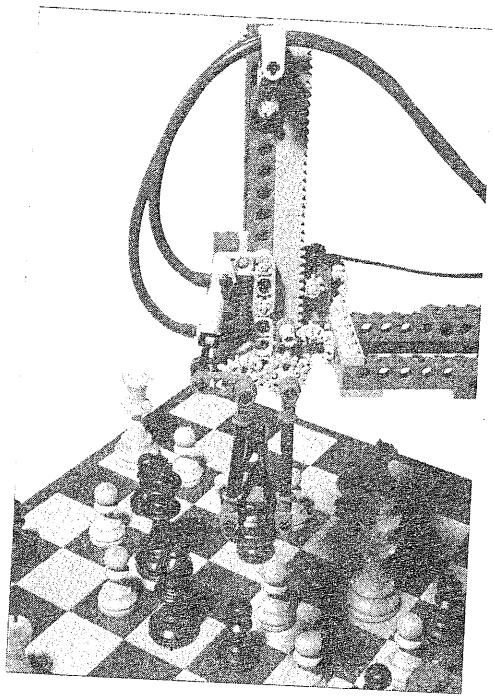


Figure 20.8 Rear View of the Lifting Mechanism

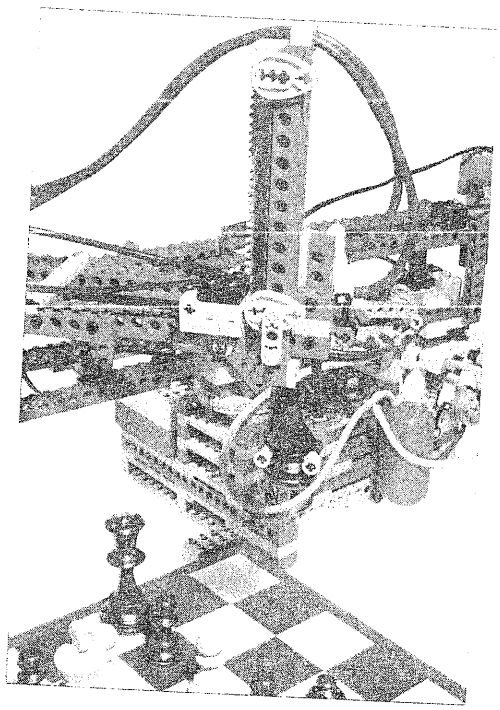
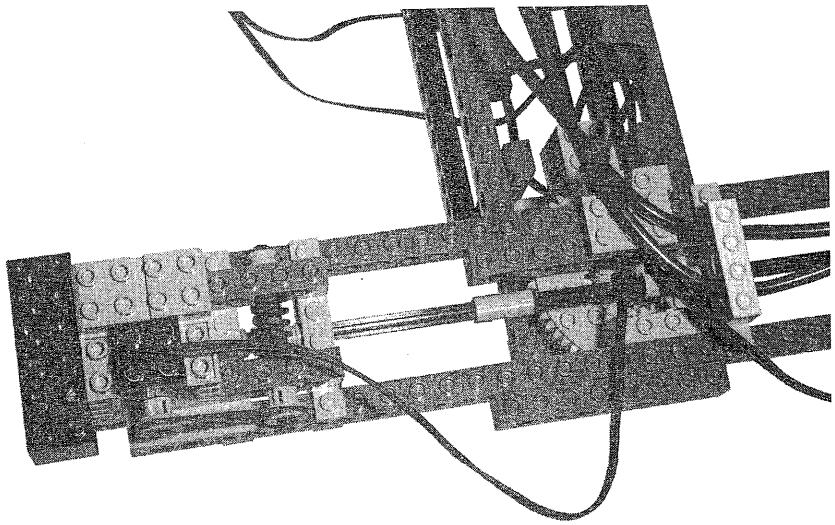
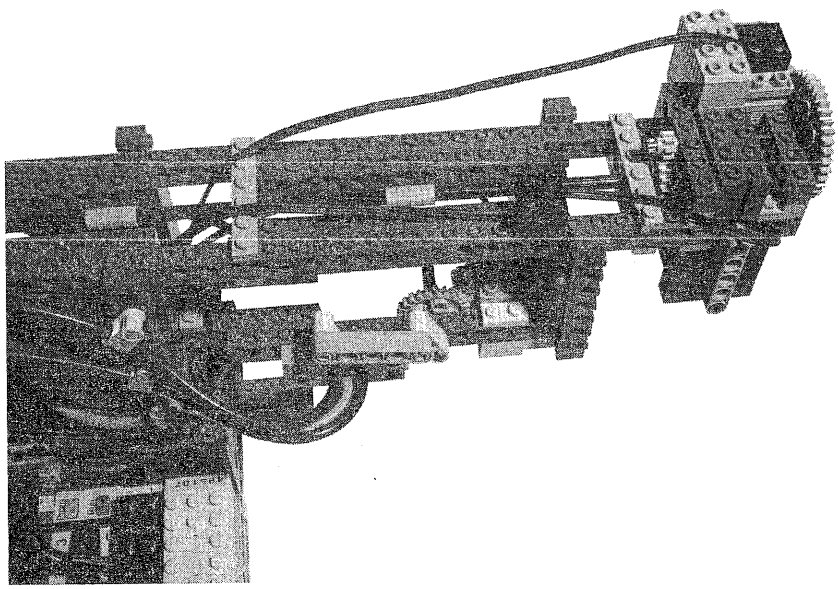


Figure 20.9 The Lifting Motor at the End of the Secondary Arm



The primary arm has a more solid structure, designed to bear its own weight plus the whole secondary arm. At its opposite end you find the motor that rotates the secondary arm, connected also to the rotation sensor that monitors that movement (Figure 20.10). The reduction geartrain is made by an 8:24 stage and a 8:40 one, for a total ratio of 1:15.

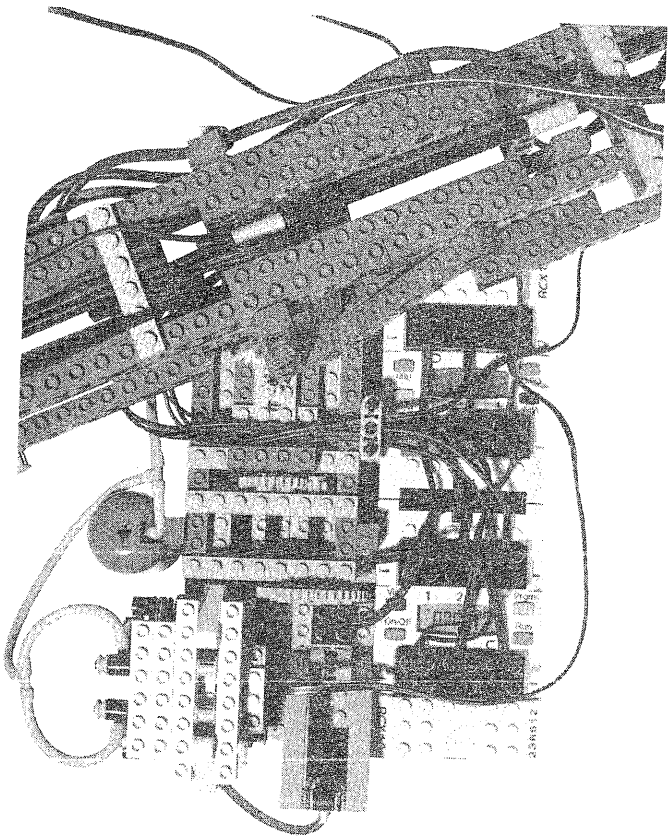
Figure 20.10 The Motors at the End of the Primary Arm



On the lower deck of the arm is the motor that operates the valve switch for the pliers. As for the secondary arm, both the motors and a couple of weight bricks serve to keep the primary arm well-balanced. If you don't do this and consequently induce a strong asymmetric load on the turntable, it may fall apart.

The base of the robot has to be very solid. It contains two RCXs, the motor that moves the primary arm and its rotation sensor, a compressor for the pliers and a pressure switch. All this weight helps in making Broad Blue very stable (Figure 20.11).

Figure 20.11 The Base



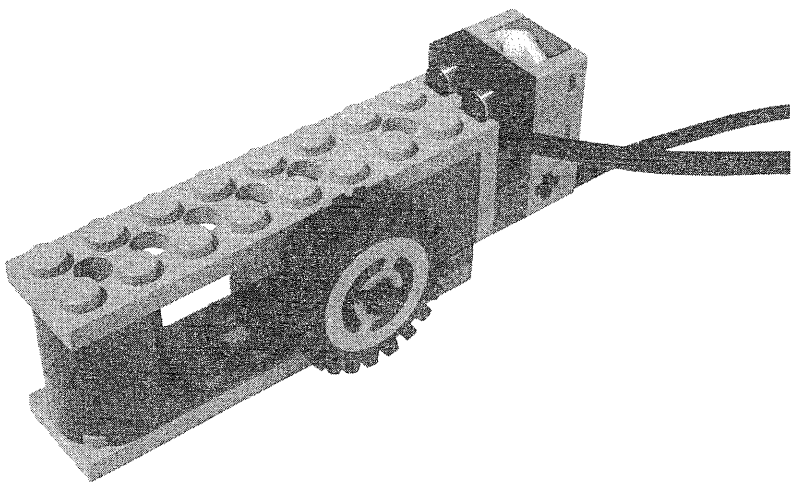
The geartrain of the primary turntable has the same configuration as the secondary one.

To provide the necessary supply of air, we used the double-acting compressor (because we had it ready), but the robot really needs a small quantity of air and any compressor should work.

It's important that you make all the wires emerge from a point close to the center of the primary turntable so they don't affect its rotation. In large robots like this, clean and well-organized wiring helps a lot in keeping things under control.

For the input of the human moves, we replaced the single touch sensor with a slightly more sophisticated unit also made from a rotation sensor. This speeds up the choice of the coordinates, changed according to rotations, displayed on the master RCX, and confirmed by a click on the touch sensor (Figure 20.12).

Figure 20.12 The Input Unit



Connecting and Programming Broad Blue

This is the only project in the book that requires two RCXs. As in most dual-RCX projects, it uses a master-slave configuration: The master decides what to do while the slave merely performs the required actions. Table 20.1 summarizes the wirings and resource allocations of Broad Blue; RCX1 is the master, and RCX2 the slave.

Table 20.1 Broad Blue Resource Allocations

Resource	Function
RCX1 IN 1	Rotation sensor of the input unit; sets user's move.
RCX1 IN 2	Touch sensor of the input unit; confirms user's move.
RCX1 OUT B	Motor that operates the pneumatic valve switch for the pliers.

Continued

Table 20.1 Continued

Resource	Function
RCX1 OUT C	Always ON, only used to supply power to compressor (via the pressure switch).
RCX1 Display	Shows the input square in the form Column.Row.
RCX1 Loudspeaker	Confirms or invalidates user's moves with an appropriate sound.
RCX2 IN 1	Rotation sensor of the primary arm; controls its position.
RCX2 IN 2	Rotation sensor of the secondary arm.
RCX2 IN 3	Touch sensor of the lifting system, closes at all up or down positions.
RCX2 OUT A	Motor of the primary arm.
RCX2 OUT B	Motor of the secondary arm.
RCX2 OUT C	Motor of the lifting system.

RCX1 receives the user's move with a protocol similar to that described for the Chess Visual Interface: starting square column, click, starting square row, click, destination square column, click, destination square row, click. At this point, it evaluates its own move, then performs it with the assistance of the slave RCX.

The communication protocol between the two programmable units is rather simple: You need 64 values corresponding to the squares, plus two more for the pliers-up and pliers-down commands. Include a "done" return value that the slave RCX uses to inform the master when it has finished the required action.

NOTE

If you want to compute the address of each square in terms of rotation sensor positions, you need some trigonometry. The nice thing is that in this case you can compute them just once on your PC, then store them in an 8 by 8 array in the program, so the RCX doesn't have to actually perform any calculation, just read the coordinates from a table.

Each turntable is like a 56t gear, and the 8t pinion acting on them results in a 1:7 ratio. The rotation sensor is placed after the first 1:3 stage, thus each complete turn of the arm corresponds to 21 turns of the sensor. The latter features 16 ticks

per turn, meaning that each turn increments a count of 336 units, corresponding to 1.07 degrees per tick.

The actual size of your chessboard, and consequently of the arms of your robot, is very critical. The longer the arms, the higher the angular resolution needed to control their movements. The arms of Broad Blue are both 22cm long from the center of their supporting turntable; this means that the end of the secondary arm covers a circle of about 138cm ($22x2x\pi$). Each degree corresponds to a sector with a circumference of 0.38cm. When the arms are aligned and aiming at the farthest squares of the board, each degree of rotation of the primary turntable acts on a circle 44cm in radius and corresponds to movement of 0.77cm at the pliers. We said that our resolution is 1.07 degrees per tick, and this results in 0.82cm in that area of the board. This is a bit critical: You might prefer to connect the rotation sensor before the first 1:5 reduction stage to increase precision by a factor of five.

The slack between gears drastically decreases the accuracy of the movements. To reduce the problem to a minimum, Broad Blue always approaches the squares from the same side of the board, going a bit farther and coming back when necessary.

Variations on the Construction

Nothing prevents you from building a Chess robot with a large XY system like the one we described for Tic-Tac-Toe. This way you can avoid the rotation sensors entirely and use the good old touch sensors and pegs technique to address all the squares.

A smaller board will result in a smaller robot, meaning less parts, less weight to support, and consequently a simpler structure. Similarly, smaller pieces will need a shorter range for the lifting mechanism.

Is it possible to build a mechanical interface similar to ours, but use only *one* RCX? We believe so. If you are able to use pneumatics for the lifting system outside that used for the pliers, you can coordinate their movement in a predefined sequence (starts with pliers up and opened):

1. Go down.
2. Close pliers (grabs the piece).
3. Go up (now the robot moves to the destination square).
4. Go down.
5. Open pliers (drops the piece).
6. Go up.

What you need is a control system that operates the valve switches in the proper sequence, a rotational or linear mechanism activated by a single motor. Can you devise one?

Applying a completely different technique, you can copy the system used in some commercial chessboards that employ an electromagnet to move magnetic pieces from the bottom. Did you ever see them in action? They drag the pieces, sliding them along the borderlines of the square, so they don't interfere with the other pieces on the board. The advantages of this technique include the fact that you don't need any lifting/grabbing system anymore, thus a single RCX can do all the work; in the way of disadvantages consider that you need many non-LEGO parts, including magnetic chess pieces, a special board, and an electromagnetic coil. LEGO doesn't produce electric coils for the RCX, but they aren't difficult to make. Be aware that they absorb much more current than motors.

Now that we think of it, you can use the sliding technique even from the top. Imagine a robot similar to ours, that instead of grabbing and lifting the piece, simply surrounds it with a sort of cage lowered from above and then pushes it along the proper path. This will save one output port, but will make the software a bit more complex because the arms cannot go straight to the destination square but rather must follow an optimal path to avoid the other pieces.

Playing Other Board Games

The techniques we described in this chapter apply to many other board games, each one having its own peculiarities and requiring some adaptations.

Checkers shares with Chess similar difficulties about moves, with the difference being that you must find a way to crown the pieces when required. Moving crowned pieces without letting them come apart might be not so easy. On the other hand, Checkers needs much simpler software than Chess.

Go, in its slightly diverse variants, is the world's most popular board game. Even though it's not played much in the West, it's widespread throughout all Asian countries. Because the pieces, called *stones*, get dropped on the board but never moved, it seems ideal for a robot equipped with an automatic dispenser. The fact that the stones are black and white also suggests the possibility of using a scanning technique similar to the one we incorporated into the Tic-Tac-Toe robot, though it would likely prove impractical on the official 19 by 19 board and even on the 13 by 13 reduced one. We recommend you limit your robot to the 5 by 5 training board. It will require more than enough effort in the way of programming. Go, in fact, is considered the most difficult game for computers to

learn how to play well—a sort of frontier for Artificial Intelligence. While the strongest Chess program defeated the human world champion, the strongest Go program cannot even beat a good amateur. A tough challenge for your RCX!

Summary

Whether or not you have an interest in board games, this chapter describes some interesting tricks that may prove useful in other situations, too. You have seen that you can input data into your system using the combined knowledge of a position and a value read from the light sensor, as we showed in the Tic-Tac-Toe robot.

The simple Chess Visual Interface taught you that even the FOS units, usually designed for decorative purposes only, may work as output devices. It also showed you that you can build a complete input interface around a single touch sensor.

Broad Blue is the most complex robot of the book. It employs some of the concepts illustrated in Chapters 10 and 11 about pneumatics, object grabbing, degrees of freedom, and others. More importantly, it demonstrates that the bounds of what can be done with the MINDSTORMS system are very far reaching.