12-1987

A Graphic Implementation of Blocks World as an Example of Artificial Intelligence

Shigenori Ochiai
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1250
A GRAPHIC IMPLEMENTATION OF BLOCKS WORLD
AS AN EXAMPLE OF ARTIFICIAL INTELLIGENCE

by

Shigenori Ochiai

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
December 1987
A GRAPHIC IMPLEMENTATION OF BLOCKS WORLD
AS AN EXAMPLE OF ARTIFICIAL INTELLIGENCE

Shigenori Ochiai, M.S.
Western Michigan University, 1987

The objective of this study is to implement blocks world as a classic example of artificial intelligence both visually and realistically.

The visual representation is implemented by using a graphic system in a microcomputer. Real world constraints are implemented as rules concerning centers of gravity.

These goals are significant in realizing principles of artificial intelligence. Visual data help a human being to quickly understand the state of a system. Also, bringing a situation closer to the real world is a step toward the final goal of artificial intelligence.
ACKNOWLEDGEMENTS

I would like to express deep appreciation to Dr. Robert Trenary for his guidance and suggestions in the preparation of this paper. Without his help, I would not have been able to complete this thesis.

Shigenori Ochiai
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
A graphic implementation of blocks world as an example of artificial intelligence

Ochiai, Shigenori, M.S.
Western Michigan University, 1987
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages ______
2. Colored illustrations, paper or print ______
3. Photographs with dark background ✓
4. Illustrations are poor copy ______
5. Pages with black marks, not original copy ✓
6. Print shows through as there is text on both sides of page ______
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements ______
9. Tightly bound copy with print lost in spine ______
10. Computer printout pages with indistinct print ______
11. Page(s) _________ lacking when material received, and not available from school or author.
12. Page(s) _________ seem to be missing in numbering only as text follows.
13. Two pages numbered _______. Text follows.
14. Curling and wrinkled pages ______
15. Dissertation contains pages with print at a slant, filmed as received ______
16. Other ___________________________ _____________________________

U-MI

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS .......................................................... ii
LIST OF TABLE ............................................................... vi
LIST OF FIGURE ............................................................ vii

CHAPTER

I. KEY CONCEPTS AND GOALS OF THE PROBLEM .................... 1
   Artificial Intelligence ............................................... 1
   Blocks World .......................................................... 2
   Goals of This Study .................................................. 3

II. BACKGROUND OF BLOCKS WORLD .................................. 6
   Previous Studies ...................................................... 6
   STRIPS ................................................................. 6
   SHRDLU ................................................................. 9
   HACKER ............................................................... 10
   NOAH ................................................................. 11
   Knowledge Representation ........................................... 15
   Logic ................................................................. 16
   Semantic Network .................................................... 17
   Frame ................................................................. 18
   Planning ............................................................... 19
   Nonhierarchical Planning ........................................... 20
   Hierarchical Planning ............................................... 21
   Summary ............................................................. 22

III. THE GRAPHICS SYSTEM ............................................. 23
Table of Contents—Continued

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Aided Design (CAD)</td>
<td>23</td>
</tr>
<tr>
<td>AutoCAD</td>
<td>24</td>
</tr>
<tr>
<td>LISP</td>
<td>26</td>
</tr>
<tr>
<td>AutoLISP</td>
<td>27</td>
</tr>
<tr>
<td>Summary</td>
<td>29</td>
</tr>
<tr>
<td>IV. PROGRAM DESIGN AND COMMANDS</td>
<td>30</td>
</tr>
<tr>
<td>Program Module Structure</td>
<td>30</td>
</tr>
<tr>
<td>START</td>
<td>31</td>
</tr>
<tr>
<td>INITIAL</td>
<td>32</td>
</tr>
<tr>
<td>IDLE</td>
<td>32</td>
</tr>
<tr>
<td>CREATE_SQUARE</td>
<td>32</td>
</tr>
<tr>
<td>MOVE_SQUARE</td>
<td>33</td>
</tr>
<tr>
<td>CHK_BFR_MOVE</td>
<td>34</td>
</tr>
<tr>
<td>CHK_AFT_MOVE</td>
<td>34</td>
</tr>
<tr>
<td>PLANNING</td>
<td>34</td>
</tr>
<tr>
<td>NORMAL_MOVE</td>
<td>35</td>
</tr>
<tr>
<td>LEFT_END_ON and RIGHT_END_ON</td>
<td>35</td>
</tr>
<tr>
<td>Program Data Structure</td>
<td>35</td>
</tr>
<tr>
<td>Property List</td>
<td>36</td>
</tr>
<tr>
<td>List for Managing Space</td>
<td>39</td>
</tr>
<tr>
<td>Commands</td>
<td>40</td>
</tr>
<tr>
<td>Initial</td>
<td>40</td>
</tr>
</tbody>
</table>

iv
Table of Contents—Continued

CHAPTER

Create ........................................ 41
Move ........................................ 41
Exit .......................................... 44
Summary .................................... 45

V. CONCLUSION AND FURTHER WORKS .............. 46

Conclusion .................................. 46
Further Works ................................ 46

APPENDICES .................................. 49

A. Hardware and Software Environment ............ 50
B. User Manual ................................ 53
C. Results .................................... 56
D. Source Program ............................ 75

BIBLIOGRAPHY ................................. 98
LIST OF TABLES

1. Elements and Functions of Logic ......................... 16
LIST OF FIGURES

1. Examples of Real World Constraints .................. 5
2. Initial and Goal States in STRIPS .................... 7
3. Process to Solve a Program in STRIPS .................. 8
4. Flow of Problem Solving in HACKER .................... 11
5. Graphic Representations of a Node .................... 12
6. An Example of a Semantic Network .................... 17
7. Inheritance Hierarchy Structure in Semantic Network ... 18
8. Structure of a Plan ................................ 20
9. Examples of Instruction ANGLE ...................... 28
10. Structure of Program in This Thesis ................. 31
11. Coordinates in the Initial State .................... 37
12. Coordinates After Creating a Square .................. 38
13. Coordinates After Creating Two Square ................ 38
14. Coordinates After Moving a Square on Another ........ 39
15. Initial State ..................................... 41
16. After Creating a Square ............................ 41
17. After Creating Three Squares ....................... 41
18. Three Points Inside a Square ........................ 42
19. Three Points Above a Square ........................ 42
20. A Square on Another ............................... 43
21. Process of Moving a Square With Space on the Table .... 43
22. Process of Moving a Square Without Space on the Table . 44
23. Examples of Realistic Situations ................... 47

vii

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER I

KEY CONCEPTS AND GOALS OF THE PROBLEM

Artificial Intelligence

Over the past few decades, many research efforts have occurred in a field that has come to be called Artificial Intelligence (AI). As a rough definition, we can say AI is the study of ideas that enable computers to be intelligent. Computer scientists have been trying to understand the principles that make intelligence possible. The meaning of "intelligence" itself, however, is ambiguous and not specified clearly. One definition of intelligence is the ability to acquire and apply knowledge; another is the ability to perceive and manipulate things in the physical world. The question of what intelligence is involves various disciplines: philosophy, psychology, biomedical science, and computer science. Philosophers ask what the essence of thinking is, psychologists study the mechanisms of intelligence, biomedical scientists research the human brain from the viewpoint of biology and medical science, and computer scientists try to implement AI.

Even though it is difficult to define intelligence, AI research is becoming more and more important. Stories on AI are regularly featured in news magazines. A main reason AI research is so important is that results of those studies exert a strong influence on daily life. In offices, computers can suggest financial
strategies and polish draft documents. On farms, computers can control pests and prune trees. In homes, computers can give advice on cooking and shopping. In factories, computers can perform dangerous and boring assembly, inspection, and maintenance jobs. The complete development of such systems is still very much a goal. However, these stories are not mere science fiction. Some of the basic ideas of AI research have already been commercialized even though some applications are not perfect and others are very simple.

The solid results in AI have accumulated because of the contribution of pioneers. Their efforts have generated several subfields: knowledge representation, search, vision, planning, problem solving, and so on. Although each subfield has difficult problems to solve, each of them contributes to another one: good knowledge representations are convenient for problems of planning, some concepts of search are useful to problem solving. In this thesis, a particular problem called blocks world is chosen to study AI.

Blocks World

Blocks world is a virtual world in the computer system. It is constructed with several labeled blocks (like children's toys) on a table or on each other, and has a hand which can pick up and move a block. There are some commands which are used to pick up an object or put one object on another. Assumptions of the blocks world are as follows: the hand can grasp only objects which do not support anything; the hand can move one object at a time; every object is
properly supported at all times; an object which is placed below
another one can be moved only after all blocks on it are removed.

Even small children can manipulate toy blocks. However, these
same tasks require many of the abilities used in solving more
intellectually demanding problems. Research on blocks world has led
to many ideas about AI: how to recognize blocks, how to represent a
situation, how to plan the way to a goal, and how to search for an
optimized solution. Also, we can use examples from this domain to
illustrate major ideas in AI.

There are many papers about blocks world. Some of them are
written from the view of "knowledge representation," some are
studied in the context of "planning." The advantage of using the
blocks worlds domain to explore AI is that the problems studied
can be highly restricted. In contrast, if knowledge representations
are attempted in the fields of poetry or writing, the difficulty of
representing knowledge in a computer is beyond imagination. If
planning is designed in a field of politics or economics, the trouble
of planning is beyond the capability of a computer. However, since
blocks world has a simple, logical, and closed domain, computer
scientists can theoretically manipulate each situation in a program.

Goals of This Study

This study has two goals: the first goal is to imitate blocks
world visually; the second one is to implement situations similar to
the real world which no other blocks world has implemented.

The first goal is important because visual data leads to quick
understanding of each situation. For example, in presentation, pie charts or histograms are frequently used because human beings can obtain more information at first sight than phrases or sentences. Even if there exists just one square on a screen, this figure can capture a lot of information: the size of the square, the location of it, and whether it is horizontally placed or rotated. When the same information is described in sentences, several lines instead of a small figure may be required.

Past studies have used only character data to represent a situation. Blocks world in this thesis is constructed in a Computer Aided Design (CAD) graphics system. Using a graphics system helps a user to understand each state better than with just character data because a drawing illustrates what happens. In addition, the system described in this study has many functions for conversational interaction. Therefore, the user can operate functions confirming the results at each step on a screen.

The second goal is significant because the similarity to the real world leads to ideas which might be useful in designing thinking machines. As the pseudo-world in the computer gets closer to the real world, the computer structure becomes more and more complicated and difficult to implement. However, AI should have the capabilities to represent and understand complex scenes because the intelligence of human beings can understand situations in the real world. AI is one kind of intelligence even though it is based on the computer system. The central goal of AI is to be able to imitate any function of human intelligence.
An aspect of the real world which has not previously been used in blocks world is the constraint imposed upon stacks of objects. In this block world the result of moving an object is similar to the real world. In the real world, it is a rare case that one block rests exactly on another one. It frequently occurs that a block is placed a little out of line to the right or left (see Figure 1).

![Figure 1. Examples of Real World Constraints](image)

When children play with blocks, it sometimes happens that a block tips over because the center of gravity of one block is outside of the block supporting it (see Figure 1). This system has capabilities to demonstrate these situations on a graphics screen and data structures to represent these states.

Attention has been focused on many aspects of blocks world research. Because of the goals described above, this study is a contribution to the study of blocks world problems.
CHAPTER XI

BACKGROUND OF BLOCKS WORLD

Review of related literature is useful to understand basic idea about blocks world. This chapter deals with the following studies: STRIPS, SHRDLU, HACKER, and NOAH. In addition, the following ideas of knowledge representation and planning are described: logic, semantic network, frame, nonhierarchical planning, and hierarchical planning.

Previous Studies

There is a great deal of research related to blocks world. Blocks world allows computer scientists to transfer complex issues to simple forms in the fields of knowledge representation and planning. Also, it has been used as a specific and constrained domain in which to explore AI issues instead of using a real and complex model.

STRIPS

STRIPS (Stanford Research Institute Problem Solver) is a problem solving system which attempts to find a sequence of operations for obtaining the goal in blocks world (Fikes & Nilsson, 1971). This problem solver represents blocks world as an arbitrary collection of first-order predicate calculus formulae. The first-order predicate calculus is a formal language in which a wide variety of statements
can be expressed. The STRIPS system uses the language of predicate calculus to express the state of a blocks world configuration. In addition, the predicate calculus is used to deduce other logical statements from a given blocks world description.

In STRIPS, each problem is a goal to be achieved by the manipulation of blocks (see Figure 2).

<table>
<thead>
<tr>
<th>Initial Operator</th>
<th>Description</th>
<th>Goal Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR(C)</td>
<td>top of C is clear</td>
<td>CLEAR(A)</td>
<td>top of A is clear</td>
</tr>
<tr>
<td>ON(C,A)</td>
<td>C is on A</td>
<td>ON(A,B)</td>
<td>A is on B</td>
</tr>
<tr>
<td>ONTABLE(A)</td>
<td>A is on TABLE</td>
<td>ON(B,C)</td>
<td>B is on C</td>
</tr>
<tr>
<td>ONTABLE(B)</td>
<td>B is on TABLE</td>
<td>ONTABLE(C)</td>
<td>C is on TABLE</td>
</tr>
<tr>
<td>CLEAR(B)</td>
<td>top of B is clear</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Figure 2. Initial and Goal States in STRIPS](image)

The solution to the problem is a sequence of operations to achieve a goal. STRIPS extracts the difference between the goal and the current state and selects a relevant operator to reduce the difference (see Figure 3). In Figure 3, operation 1 is generated to reduce the difference between a part of the initial conditions, CLEAR(C) and ON(C,A), and a part of the goal, ONTABLE(C). The second state satisfies one condition of the goal conditions, ONTABLE(C). Next the program notices ONTABLE(B) and CLEAR(B) in the second state and ON(B,C) in the goal. By operation 2, conditions ONTABLE(B) and
CLEAR(B) transfer to ON(B,C). The last unsatisfied condition is ON(A,B). Operation 3 lets the condition ONTABLE(A) change to ON(A,B). The fourth state is exactly the same as the goal. In other words, STRIPS is in the final state which means there is no difference between the goal and the current state (i.e., the current state is the goal).

Figure 3. Process to Solve a Problem in STRIPS
Winograd (1972) developed a system named SHRDLU, (it may be RSHDLU in 1987 (Chiles, 1987)), in order to attempt understanding natural language in blocks world. One reason that blocks world is used is that commands are in the form of very simple statements instead of complex phrases.

The system contains four basic elements: a parser, a recognition grammar of English, programs for semantic analysis, and a general problem solver. In this thesis the main focus is on how to implement something like the problem solver in SHRDLU and this does not deal with the other parts of the SHRDLU system.

SHRDLU translates knowledge about the state of the world into MICRO-PLANNER assertions, and embodies reasoning knowledge in MICRO-PLANNER programs. MICRO-PLANNER is a programming language and an abbreviated version of the original PLANNER language. It is written in LISP. An input sentence "A square is on the table" might be translated into an assertion of the form:

\[(ON \text{ SQUARE } \text{ TABLE})\]

For example, a problem to put one block on another might be a MICRO-PLANNER program such as the following:

\[
\begin{align*}
\text{(THGOAL} & \text{(ON } ?X ?Y) \\
& \text{(OR} (\text{ON-TOP} ?X ?Y) \\
& \text{(AND} (\text{CLEAR-TOP} ?X) \\
& \text{(CLEAR-TOP} ?Y) \\
& \text{(PUT-ON} ?X ?Y)))
\end{align*}
\]

THGOAL is the name of this function and this function represents a relation ON which has two variables, X and Y. THGOAL means to prove X is ON Y. At first (ON-TOP ?X ?Y) checks whether X is already on Y.
or not. If it is not, the following three functions are achieved:
1) clearing off everything that is stacked on top of X, 2) clearing off Y, and 3) then putting X on Y.

In MICRO-PLANNER a procedure is a program, and its operations carry out actions. The THGOAL procedure finds an assertion in the database or proves it with other procedures. It is advantageous to solve problems without checking all entities in the database as a blind theorem prover might do.

**HACKER**

HACKER was developed as a computer model of skill acquisition by Sussman (1975). The strategy of HACKER is that the problem solver designs a new procedure to obtain the final goal when the answer library does not include a procedure to solve a problem. When the same problem occurs, a plan to solve the problem has already existed in the answer library.

HACKER has two main subsystems, the answer library and debugger (Figure 4). The answer library has a set of schemes that describe possible subgoal interactions. The program tries to match schemes of the answer library to the goal structure of the current plan. If possible plans exist, the planner proposes them. If the answer library does not have any plan to obtain the goal, the debugging procedure corrects errors in the plan and stores new data in the answer library to avoid the same process. HACKER has the advantage of correcting errors in a plan with debugging programs.
Traditionally backtracking methods blindly try another problem solving operation when a sequence fails. As the databases grew larger, an effective and efficient plan should be considered. HACKER is an example of avoiding wasteful iterations occurring in the backtrack approach.

**NOAH**

NOAH (Nets Of Action Hierarchies) has a representation for plans called the procedural network (Sacerdoti, 1977). This mechanism generates a plan hierarchy as opposed to a single-level plan. Hierarchical planning solves interaction problems which arise when a planner solves problems using an arbitrary order for achieving a goal. Operators are not ordered until a potential interaction is detected, and then they are ordered to avoid the interaction. Planners in previous research ordered operations arbitrarily, and, if an interaction emerged, they backtracked and replanned to try to avoid the interaction. NOAH does not need to backtrack and can
develop a plan more effectively.

A procedural network, also known as a procedure net, is a strongly connected network of nodes, each of which may contain both procedural and declarative information. The procedural knowledge is used for generating more detailed subactions at levels of greater details. The declarative knowledge is used to model the action at the node's own level of detail. For example, if a procedure is executed that puts one block on top of another, the procedural knowledge expands the goal into subgoals to clear the top of both blocks and put one block on another. Also, NOAH records that the supporting block no longer has a clear top surface. Because the declarative knowledge represents the effects of actions, the system can reason with them.

The procedure net is a graph structure. An action at a particular level is represented by a single node in the procedure net (see Figure 5). Each node contains procedural information, declarative information, and pointers to other nodes. The nodes are linked to form hierarchical descriptions of operations, and to form plans of action. Nodes at each level of the detail hierarchy are linked in a partially ordered time sequence by predecessor and successor links. Each sequence represents a plan at a particular level of detail.

![Figure 5. Graphic Representations of a Node](image)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
As an example, the problem in Figure 3 is examined. NOAH uses a list of the task-specific SOUP (Semantics Of User's Problem) functions, in this case CLEAR and PUTON. At first, the system builds an initial procedural net with a single node:

**LEVEL 1**

| Achieve (AND(ON A B)(ON B C)) |

Level 1 is split up, so that each of two actions is achieved independently (Level 2).

**LEVEL 2**

```
S       Achieve (ON A B)       J
       Achieve (ON B C)       
```

However, these operations are not immediately invoked, instead waiting for a reason to do so. The PUTON functions are expanded and the result is the following:

**LEVEL 3 (First Step)**

```
S       Clear A             J
        Clear B            1
2       Put A on B

S       Clear B             J
        Clear C            4
5       Put B on C
```

In this step, node 3 will delete node 4 (B is clear), because node 3 denies CLEARTOP B. NOAH notices that CLEARTOP B is implemented in the effects of both nodes 4 and 6. Since NOAH has not achieved any of its goals in a particular order, it need not backtrack to modify its plan. Instead, it uses this conflict as an opportunity to introduce constructively a partial ordering of goals: it decides to accomplish node 3 after it has done everything else.
LEVEL 3 (Second Step)

Next the system observes that nodes 2 and 4 are redundant and eliminates node 2.

LEVEL 3 (Final Step)

Then, NOAH expands the CLEAR A goal at level 3. To achieve CLEAR A, the program needs to move C off of it and put C someplace; it does not matter where. Block C cannot be moved unless it is clear, so the final sequence to clear A is CLEAR A and Put C on OBJECT 1.

LEVEL 4 (First Step)

The planner notices that node 6 may interfere with its latest goal, so decides to order node 6 after it has achieved Put C on Object 1.
Finally, the system finds that CLEAR C is mentioned twice in the plan, therefore, it eliminates one of the nodes. The final plan is:

LEVEL 4 (Final Step)

NOAH needs to have several steps for obtaining the final plan. However, it is not required to replan and waste time and space. Therefore, it can construct the plan effectively and efficiently.

Knowledge Representation

Human beings can obtain knowledge as soon as they see a scene. If there is a set of blocks on a table, a person immediately understands such a situation: one block rests on another, a block is between two blocks, and so on. However, in a computer program it is difficult to express states which human beings can comprehend without any trouble. Consequently, it is important to use a good data structure to express knowledge because designing a good representation is often the key to turning hard problems into simple ones. Although there are many ways to represent knowledge, this section presents three approaches described as "logic," "semantic network," and "frame."
Logic

One approach to representing knowledge is formal logic as a calculus of the process of making inference from facts. The STRIPS system described in this chapter uses this concept. Elements to construct logic are NOT, AND, OR, IMPLIES, and EQUIVALENT (Table 1). The meaning of each symbol is the same as in Boolean algebra. For example NOT A is the complement of A and when A is true (T), NOT A is false (F). A AND B is true (T) if and only if both A and B are true (T).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>NOT A</th>
<th>A AND B</th>
<th>A IMPLIES B</th>
<th>A EQUIVALENT B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Note: T true  F false

Logic based knowledge is convenient to use because after obtaining a logic expression from the problem it is easy to express it in computer architecture based on Boolean algebra.

Logic can organize the representation scheme by building up from a simple notion (like that of truth and falsehood) to create complex notions (like that of conjunction and predication). This approach mechanizes the derivation from initial states to the goal. When the source and rules are defined, the destination is determined automatically. However, when the number of facts becomes large,
there is a combinatorial explosion of the possibilities at each step. For example, natural language understanding makes the program handle a huge database. It is difficult to use logic for implementing a system which must manipulate an enormous amount of information.

**Semantic Network**

The semantic network, developed by Quillian (1968) and others, models the human associative memory. The phrase "semantic net" is sometimes used for "semantic network." Not only computer scientists but also philosophers, psychologists, and linguists are interested in the human semantic memory because this model relates to the question of what semantic information is and how a person's brain stores and uses such information. In other words, a semantic network is a model which tries to illustrate the mechanism of a human brain's memory as a graph.

![Semantic Network Diagram](image)

**Figure 6. An Example of a Semantic Network**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
A network consists of nodes and links between nodes. Nodes express objects, events, and concepts and a link represents the relation between nodes. Figure 6 illustrates a semantic network in which labeled circles are nodes and labeled arrows are links.

Figure 7 demonstrates a semantic network representing a syllogism. This provides one explanation for the popularity of semantic networks as a knowledge representation. Two facts linked by "is-a," "Mary is a girl" and "A girl is a human being" are used to deduce a third fact, "Mary is a human being." In comparison with logic, an advantage of a semantic network is that relevant facts can be inferred without a search through a large database.

![Figure 7. Inheritance Hierarchy Structure in Semantic Network Frame](image)

An extended and sophisticated model of a semantic net is the frame. A frame is a powerful data structure for representing stereotyped situations. The concept of frames was originally proposed by Minsky (1975). A frame provides a way to denote objects and describe the relations that hold among them. The frame is a promising idea, still in its early development stage. An advantage of the frame is that it proposes ways for linking fragments of knowledge to decide how they are to be used. Knowledge will be used when it is appropriate and not used when it is inappropriate.

A frame is constructed from some number of "slots" that together
describe a stereotyped object, act, or event. Putting it all together, a list is:

( <frame name> ( <slot 1> <value 1>  
<value 2>  
.  
.  
)  
( <slot 2> <value 1> ...) ...)  
.  
.  
)

The following are examples:

(HENRY (A-KIND-OF MAN)  
(HEIGHT 1.78)  
(WEIGHT 75)  
(HOBBIES (JOGGING SKIING))  
(OCCUPATION (TEACHING RESEARCH)))

(SQUARE1 (LOCATION (7.0 2.5))  
(ON TABLE))  
(SIZE (1.0 0.5)))

The concept of frames makes it possible to summarize complicated descriptions by recognizing patterns called abstraction units. It is an established fact that abstraction is important for conveying information rapidly. Therefore, frames are useful in manipulating information in a computer system.

Planning

Planning means deciding on a course of action before acting. How to make a cup of coffee is a plan. Almost all plans have small subplans which organize a plan. Each goal of a main plan can be replaced by a more detailed subplan to achieve it. For example, a main plan, making a cup of coffee, needs subplans: boiling water,
getting coffee beans, choosing a cup, and so on. If coffee beans are unavailable, another subplan which requires a trip to the store to buy coffee beans occurs. If you have no money, you go to the bank to get money before going to the store. A plan illustrates the structure of subplans (see Figure 8). There is a waste of effort when a plan is too concrete. Too detailed a plan can be difficult to use when it creates high level strategy. On the other hand, a program cannot manipulate each state of planning if the plan is too abstract.

![Figure 8. Structure of a Plan](image)

Too symbolic a plan is difficult to implement in a program. A balance between abstract and concrete information is necessary for efficient planning.

This section discusses two approaches to planning, nonhierarchical planning and hierarchical planning.

**Nonhierarchical Planning**

Nonhierarchical planning means that a planner has only one representation of a plan. STRIPS and HACKER described in this chapter are examples of this type of planning. A nonhierarchical planner develops a sequence of problem solving actions to achieve its
goals. This approach reduces the difference between the current state and the goal and is in the final state when there is not any difference between the current state and goal (see Figure 3).

Nonhierarchical planning must trace all possibilities of planning for getting the solution even if a sequence fails after wasting search space and time. This approach has a major disadvantage because it can get bogged down in unimportant and unnecessary details although the strategy is simple and easy.

Hierarchical Planning

Hierarchical planning utilizes a hierarchy of representations of the plan. NOAH described in this chapter is an example of this type of planning.

This method first sketches a plan, then refines the vague parts of the plan into more detailed subplans. Finally a plan transfers to a complete sequence of operations to solve the problem. The goal of this approach is to develop a plan at a level where the details are not ambiguous. By this strategy a planner can avoid wasting search space and time and accomplish the goal efficiently and effectively. Hierarchical planning determines which subgoals are critical to the success of the plan and ignores all other possibilities.

It is advantageous that a problem solver obtains an efficient and effective final plan without backtracking and replanning, though a planner must implement a complicated strategy.
Knowledge representation and planning are important concepts in AI research.

Logic is an approach to representing knowledge and a calculus of the process of making inference from facts. STRIPS is based on logic. Another way is semantic network as a model of the human associative memory. A frame is a sophisticated extension of a semantic network.

Two approaches to planning are nonhierarchical planning and hierarchical planning. Nonhierarchical planning means that a planner has only one representation of a plan. Examples of this concept are STRIPS and HACKER. Hierarchical planning utilizes a hierarchical representation of the plan. NOAH has this character.
CHAPTER III

THE GRAPHICS SYSTEM

To familiarize the reader with the graphic system in this study, this chapter explains a computer aided design (CAD) system named AutoCAD and a good tool called AutoLISP to explore blocks world.

Computer Aided Design (CAD)

Computer graphics are used today in many different areas of industry, business, and education. In business, management information systems require graphics terminals for displaying charts. In education, teachers use graphics to advantage in many ways. For example, the computer language LOGO uses graphics to teach programming without requiring a complicated syntax in the programming language.

In industry, Computer Aided Design (CAD) is an example of computer graphics. The purpose of CAD is to lighten the burden of the designer. In CAD, interactive graphics are used to design components and systems of mechanical, electrical, and electromechanical devices. It is useful for a designer to draw a blueprint plan without papers and pens, because one is not ever faced with a dirty drawing, thus reducing the number of mistakes. In addition, interactive interfaces make a designer check the details of a plan visually. As some CAD systems have functions to display a
drawing with several colors, a designer can understand it quickly.

It was once true that computer systems and graphics devices (i.e., graphics displays, plotter, and graphics memories) were expensive and people could not obtain a CAD system easily. However, technological improvements make it possible for customers to use CAD systems inexpensively. Although in the past CAD was used only on mainframe systems, today, even a microcomputer can support CAD. CAD systems are becoming more popular and widely used.

AutoCAD

AutoCAD is a CAD system provided by Autodesk Inc., which runs under the PC-DOS or MS-DOS operating system using an IBM PC or IBM PC compatible machine with a graphics board.

With AutoCAD the user can easily draw many kinds of diagrams such as architectural blueprints or electric circuit diagrams by typing commands and pointing to positions on the screen. The user does not need to learn a special programming language. AutoCAD provides many commands to make creating a figure an easy task, such as "LINE," "CIRCLE," etc. This system has a comfortable user interface. Several commands provide options and help. For example, after typing the command "CIRCLE," the user can select either "center point and radius" or "three points" to specify the circle location. Positioning is accomplished by using a cross hair cursor. If one makes a mistake, the user can recover by using only one command, "ERASE." When the user repeatedly draws the same entities (e.g., circle, triangle, or rectangle), commands "BLOCK" and "INSERT" are
very helpful. These commands allow a user to name the entry and recall it by the name. Those commands are similar to macro statements in an assembly language.

Convenient input and output devices are available. The user can use a mouse, tablet, or trackball as the input device on AutoCAD. These devices allow the user to move a cross hair cursor quickly. Also, a pen plotter or printer plotter can be connected to the system as the output device. The user can print the diagram by selecting a command in a menu. With a pen plotter or printer plotter colored diagrams are obtained. In addition, AutoCAD allows the output to be formatted for different sizes and orientations.

Editing (e.g., move, copy, or erase) is easy. The user can indicate either an object or an area by using a "window" framed with the pointing device. When the user uses the command "MOVE," one can select "DRAG" mode. Dragging makes the user understand which entity one is moving. The command "COPY" has a function to make multiple copies. Also, the command "ARRAY" is similar to the "COPY" command. This command not only allows the user to make multiple copies, but also enable copies of selected objects to be arranged in a rectangle or polar (circle) pattern. By the command "ERASE" the user can erase entities easily. Even if the user inadvertently erases the entity which one does not want to erase, "OOPS" command allows the user to recover it. These commands let the user edit a drawing. After drawing the diagram, it can be saved on secondary storage, and the user can retrieve it again easily.

AutoCAD is a powerful system for graphic design. It is a main
advantage that the user can use it without any knowledge about a special language. Each command is understandable and easily used. Consequently, AutoCAD is useful to implement the visual blocks world.

LISP

LISP is a programming language that takes its name from the phrase LIST Programming. LISP is a very popular and powerful tool in AI research.

The basic building blocks of LISP are atoms and lists. An atom is either a number or a string of alphanumeric characters. Atoms are sometimes used for such things as relationships, properties, and specific objects. A list is a basic data structure and an ordered set of elements (either atoms or other lists).

LISP has many advantages which other languages, PASCAL, FORTRAN, COBOL etc. do not have. One advantage is the ability to easily manipulate lists. List processing is widely used in the field of knowledge engineering.

Another merit of LISP is that it is a functional language. A programmer is allowed more abstract and symbolic programming. A designer does not need to be concerned with details of syntax or the declaration of variables or data types. As all functions have a return value, a designer assumes procedures and data have the same form. The difference between procedure and function is that a function returns a result while a procedure assigns a value to one of declared parameters. In LISP one program can use another as data because LISP's data structures are all built from list structures.
Another advantage of LISP is its manipulation of property of objects. It is easy not only to construct lists to represent properties, but also to program the properties by using the instruction "ASSOC." The instruction "ASSOC" needs two arguments: the key specified by the first argument and the association list specified by the second argument. ASSOC moves down the association list until it finds a subset equal to the key. The return value of ASSOC is the entire element (i.e., key and associated data) or NIL if the key is not found. For example, assume the name of a list is BLOCK-A and the contents of the list is the following:

```
((COLOR RED) (SUPPORTED-BY BLOCK-B) (IS-A BLOCK))
```

ASSOC would produce the following examples:

```
(ASSOC 'COLOR BLOCK-A) return value: (COLOR RED)
(ASSOC 'IS-A BLOCK-A) return value: (IS-A BLOCK)
(ASSOC 'SIZE BLOCK-A) return value: NIL
```

Although LISP has many dialects including MacLISP, InterLISP, ZetalISP, FrantzLISP, CommonLISP, and so on, basic concepts are the same in all of these LISP variants.

AutoLISP

AutoCAD supplies a programming interface called AutoLISP for creating a user's utilities. AutoLISP is similar to the programming language LISP, and has almost all typical LISP functions.

Although AutoLISP resembles other LISPs, it has many other functions to handle graphic data (e.g., location, distance, angle, etc.) are added, for example, the function "getpoint" is used to manipulate the location on the screen and to pause for user input of
a point. When the user indicates a point, this function returns the location of a point. Another function "distance" allows the program to calculate the distance between two points. The function "angle" is similar to "distance" in the sense of using angle instead of distance. This function returns the angle in radius between two points. For example, when two points define a horizontal line (i.e., the value of y coordinate is the same) the value 3.141593 (i.e., 180 degree) is returned (see Figure 9 (a)). The example in Figure 9 (b) illustrates the return value of 45 degree.

![Figure 9. Examples of Instruction ANGLE](image)

There are many functions for handling graphics data in addition to the above functions. Those functions are not usually supplied in LISP. By using those functions AutoLISP can manipulate graphics data.

On the other hand, AutoLISP does not have many capabilities to describe a relation between objects as a usual LISP. For example, it lacks the command "GET" to get the property of an object in a list. This command is very useful for dealing with a relation between objects. However, AutoLISP can define the command "GET" as a customized function.

One reason that AutoCAD adopts AutoLISP is to allow a user to
easily create a new command. If the user has a customized function, the command can be used to draw a diagram quickly, because the user can obtain the same drawing by typing only one command instead of several ones. When the user frequently uses a same size rectangle, it is useful to define a customized function, for example, named "BOX." Assume the function "BOX" can draw a rectangle indicating only two positions: lower left and upper right corner of the rectangle instead of indicating the four corners of a rectangle.

AutoLISP is a good tool to research blocks world since it has the capabilities for describing relationships between objects and manipulating graphics data. Also, it is easy to construct commands, because AutoLISP has the ability to create customized functions. AutoLISP can define many functions to express properties of objects. Therefore, it was chosen to use for researching blocks world.

Summary

In this thesis, a graphics system named AutoCAD is used to implement blocks world. This system on a microcomputer, IBM-PC, has many convenient functions for a user. AutoLISP is one dialect of LISP and provides an interface between the AutoCAD system and a user.
CHAPTER IV

PROGRAM DESIGN AND COMMANDS

This chapter illustrates the function and organization of the following program modules: START, INITIAL, IDLE, CREATE_SQUARE, MOVE_SQUARE, CHK_BFR_MOVE, CHK_AFT_MOVE, PLANNING, NRML_MOVE, RIGHT_END_ON, and LEFT_END_ON. In the second section, the data structure is mentioned. In the last section, the usage of each command and results on the screen are described.

Logic, semantic network, and frame, are explained to represent knowledge as described in chapter II. In this thesis, the frame concept is used in a data structure named "property-list." Also, nonhierarchy planning is modeled to implement a plan.

In the last section, a situation in which a square inclines in the case of the center of gravity outside of the target one is illustrated as an example of a particular problem which happens in a general nature, though previous studies have not implemented it. In addition, planning is carried out when a square is below another one.

Program Module Structure

The program of this thesis is written in AutoLISP. Figure 10 illustrates the structure of this program. Arrows indicate that one part of the program might call another directly. The top level
function START calls INITIAL and IDLE. After INITIAL ends, IDLE starts and does not stop until a command, "exit," is typed. IDLE calls CREATE_SQUARE or MOVE_SQUARE by the user's typing command. MOVE_SQUARE calls CHK_BFR_MOVE (check before move) at first. It depends on the return value of CHK_BFR_MOVE whether MOVE_SQUARE calls CHK_AFT_MOVE. CHK_AFT_MOVE (check after move) calls PLANNING when it is necessary. Then, according to the situation, CHK_AFT_MOVE calls one of three functions, NRML_MOVE (normal move), RIGHT_END_ON, or LEFT_END_ON, or does not call any function.

START

START

START

START

INITIAL

initial

initial

initial

CREATE_SQUARE

CREATE_SQUARE

CREATE_SQUARE

MOVE_SQUARE

MOVE_SQUARE

MOVE_SQUARE

CHK_BFR_MOVE

CHK_BFR_MOVE

CHK_BFR_MOVE

CHK_AFT_MOVE

CHK_AFT_MOVE

CHK_AFT_MOVE

PLANNING

PLANNING

PLANNING

NRML_MOVE

NRML_MOVE

NRML_MOVE

RIGHT_END_ON

RIGHT_END_ON

RIGHT_END_ON

LEFT_END_ON

LEFT_END_ON

LEFT_END_ON

Figure 10. Structure of Program in This Thesis

START

This function starts a program to create and move a square in blocks world. A user should invoke the function named START at the beginning. After starting the program, a user does not need to let this function run. This function calls two functions INITIAL and IDLE.
INITIAL

A function INITIAL sets initial values for variables. This function displays a message, "When you want to create a square, type create" to notify the user of the program's start. This function is called only one time from the START routine.

IDLE

This function waits for a user to type input data and invokes an appropriate function. IDLE calls a subroutine CREATE_SQUARE or MOVE_SQUARE deciding upon one of them by a string which is typed by a user. At first IDLE obtains a string from the keyboard. Then this function checks to see if the string is "create," "move," or "exit." When a string matches "create," IDLE calls CREATE_SQUARE. When a string matches "move," it calls MOVE_SQUARE. When a string matches "exit," the program terminates. If a string does not match one of the above three words, this subroutine answers to a user "I'm sorry. I do not understand what you want to do. Please type exactly again." Then this function returns to the beginning again and waits until the user types in a response to the message, "create, move, or exit?"

CREATE_SQUARE

This function draws a square on the screen and modifies lists in the program. At first, this function calls another subroutine named CHECK_TABLE. If there is any space on the table, CHECK_TABLE modifies a list for managing space to create a square and returns true. If CHECK_TABLE returns a true value, this subroutine asks the
user "Please give me the name of the square." If the same name
already exists as user's indicating, a message, "That name already
exists. Please consider another name." appears on the screen.
CHECK_TABLE returns nil when a table in blocks world does not have
any space to create a square. When CHECK_TABLE returns nil, a
function CREATE_SQUARE answers "You can not create a square because
of filling the table." Also, this function retains the current
drawing and lists.

MOVE_SQUARE

This function moves a square indicated by the user from the
current location to the new one. After this function asks the user
"Please pick up the square which you want to move," it calls another
function CHK_BFR_MOVE. CHK_BFR_MOVE returns true if there exists a
square including the indicated location by the user. Otherwise, it
returns nil. If MOVE_SQUARE obtains a true value from CHK_BFR_MOVE,
it asks the user "Please indicate the point where you want to move
the square." Otherwise, MOVE_SQUARE displays a message "I'm sorry. I
cannot find the square which you have indicated." MOVE_SQUARE calls
CHK_AFT_MOVE, when CHK_BFR_MOVE returns true and obtains a new
location. CHK_AFT_MOVE examines a new location to see if it is
appropriate or not. If a new location is appropriate, CHK_AFT_MOVE
moves a square and modifies lists. Otherwise, it does not change the
current drawing and lists.
**CHK_BFR_MOVE (check before move)**

This function checks which square the user has indicated. If CHK_BFR_MOVE finds a square including the location, it sets the property values of the square to variables. If this function does not find any square, it returns nil.

**CHK_AFT_MOVE (check after move)**

This function, at first, checks whether a moving square is below another square or not. If a moving square is below another one, CHK_AFT_MOVE calls another function PLANNING. Then this function searches whether a new location rests on another square or out of the right or left edge of the target square. If new location rests on another square, CHK_AFT_MOVE calls another function NORMAL_MOVE. If the center of gravity of the moving square is out of the edge of target square but the left or right end of the moving square is on the target square, CHK_AFT_MOVE calls a function LEFT_END_ON or RIGHT_END_ON. If the new position is below the table or out of the right or left edge of the table, this function warns the user a message, "I'm sorry. I cannot place it on that point" or "Your square will drop out from the table. I cannot place it on that point."

**PLANNING**

This function moves one or more squares to an appropriate location but not on the target square. At first, this function tries to find any space on the table to place the extra square, but not on the moving square nor target square. If there exists any space on
the table, this function moves these squares to that place. If there is not any space, this function moves the squares onto other squares, other than the moving and target square. PLANNING calls BFR_GOAL_MOVE to move another square to another place except neither on to the moving square nor the target square. Finally, this function modifies lists.

NORMAL_MOVE

"Normal Move" means that a moving square can be placed on a target square. This function draws the moving square at the new position and modifies the property list and space managing list. This subroutine searches the target square.

LEFT_END_ON and RIGHT_END_ON

These functions are the same function except the edge is right or left. One of these functions is used to draw the square at a new position instead of using NORMAL_MOVE when the center of gravity is outside of the target square. These functions move a square to a new location and obtain an angle of declivity or acclivity. Finally, these functions call another function DECLINE_DRAW to draw a declining square.

Program Data Structure

This program has two types of data structures. One of them is a list which represents the properties of each object using the frame concept (Minsky, 1974) and the other one is a list for managing
space when this program creates or moves a square in the blocks world.

Property List

One of the lists is named "square-list" and is used to represent what property each object has: on the table or below another object, the name of the object given by a user, the location of the center of gravity, and the height and width of the object. The program changes the contents of square-list when displacement of a square from one location to another has occurred.

For example, in the initial state (see Figure 11), the square-list is nil. When a state has only one square (see Figure 12), that list is the following:

```
((A ((location (7.0 2.5)) (on table) (size (1.0 0.5)))))
```

This list expresses that the square's name is "A," the center of gravity of this square locates at the coordinate (7.0, 2.5), the relationship with table is "on table," and the half width and half height of the square are 1.0 and 0.5. After another square named "B" is created (see Figure 13), this list is changed into the following:

```
((A ((location (7.0 2.5)) (on table) (size (1.0 0.5)))))
(B ((location (3.5 2.5)) (on table) (size (1.0 0.5)))))
```

This new list means that another square named "B" is added and that its location is at coordinate (3.5, 2.5). The relationship and size of the new square are the same as the old one. When the user moves a square to another position, this square-list changes. Assume that square "B" moved on top of the square "A" (see Figure 14). This square-list changes the following:
Since the square "B" moved on top of the square "A," the status of each square changes. The square "A" has relationships with not only the square "B" but also the table. The relationship between the square "A" and "B" is "below B." Also, square "B" has a new relationship with square "A" which is "on A." The square "B" has a new location at the coordinate (7.5, 3.5).

When the user indicates a new position using the mouse, the center of gravity of the square moves to the new position.

The other properties in the list are height and width. The height and width of the object would be useful when commands "ENLARGE" or "SHRINK" are implemented.

When a new location is not correct (i.e., a user indicates under the table or right or left over the table) the property list is not changed and the drawing remains unchanged.

Figure 11. Coordinates in the Initial State
Figure 12. Coordinates After Creating a Square

Figure 13. Coordinates After Creating Two Squares
Figure 14. Coordinates After Moving a Square on Another

List for Managing Space

The other list used by the program is "space-list." It expresses how much and between what points there is room to create the square on the table. Before a square is created (see Figure 11), this list has the default value:

\(((12 \ (1.0 \ 13.0)))\)

The number "12" means that this table has the space of the size "12." The next element \((1.0 \ 13.0)\) tells that size "12" begins at the x coordinate 1.0 and ends at the x coordinate 13.0. When a new square is created (see Figure 12), this list is changed to the following:

\(((5.0 \ (1.0 \ 6.0)) \ (5.0 \ (8.0 \ 13.0)))\)

After creating a new square, the space on the table is divided into two parts. The first part has the size 5.0 with start and end
positions, 1.0 and 6.0. The second part has the same size as the first part. However, the start and end position are 8.0 and 13.0. When another new square is created (see Figure 13), the size is divided into three parts:

\[(1.5 (1.0 3.5)) (1.5 (4.5 6.0)) (5.0 (8.0 13.0))\]

The first and second part have the same size 1.5 and different start and end positions, (1.0 3.5) and (4.5 6.0). The third part has a size different than the first two parts. The size is 5.0 and the start and end positions are (8.0 13.0). After moving a square (see Figure 14), the space-list is expanded. Although the second part is the same, the first part is transformed from two small parts to one big part. By this change, the program can understand that a new square can be created at both of those place.

When all available spaces of this list are smaller than the default width of the square, the system warns to a user, "You can not create a square because there is no space on the table."

Commands

This Section states how each command (create, move and exit) is used and what situation appears as a result.

Initial

In the initial state (Figure 15), the system asks "create, move, or exit?" Then the system waits for response from a user.
Create

The system provides the "create" command to create a square. When a user types "create," the system asks "Please give the name of the square." After a user names a square, the system draws a square on the table (Figure 16).

A

Figure 16. After Creating a Square

A user does not need to indicate the size and location of the square. The square name is displayed inside the square. If the name of the square is duplicated, the system warns a user "That name already exists. Please consider another name."

Move

After creating some squares (Figure 17), a user can move a square. When a user types "move," the system asks "Please pick up the square which you want to move."

B A C

Figure 17. After Creating Three Squares
A positioning device "mouse" is available to indicate the location on the screen. When a user indicates any position inside a square (X, Y, or Z in Figure 18), the system can automatically find which square to pick up.

![Figure 18. Three Points Inside a Square](image)

Next, the system displays the message "Please indicate the point where you want to move the square." A user does not require positioning exactly. Even if a user indicates a higher position than the square (point U or V in Figure 19), the system assumes that the position is on the square (point W in Figure 19).

![Figure 19. Three Points Above a Square](image)

When a new location is inside of the object, the square is placed on the object (Figure 20 (a)). If the center of gravity is outside of the object, the square would incline (Figure 20 (b)).

After repeating "create" and "move," a situation could happen in which many accumulated squares are on top of one another (Figure
21 (a)). In Figure 21 (a), when a user would move square "B" onto square "F," the system can accept such an operation.

At first, the system moves the top square "E" to a space on the table (Figure 21 (b)). All squares on "B" are moved to another place (Figure 21 (c)). Then the square "B" is moved onto the square "F" (Figure 21 (d)).
If there is not any room on the table before moving a square (Figure 22 (a)), the system places squares (C-E), except the target square (B), on another one (G). Then target square (B in Figure 22 (a)) is moved to the new location. If the new location is inside of the object square, the square is placed on the object square (Figure 22 (c)). If the new location is outside of the square, the square is inclined (Figure 22 (d)).

![Figure 22](image-url)

**Figure 22. Process of Moving a Square Without Space on the Table**

**Exit**

A user can terminate blocks world by using the command "exit."
Summary

This study has a modularized program. Modules are START, INITIAL, IDLE, CREATE_SQUARE, MOVE_SQUARE, CHK_BFR_MOVE, CHK_AFT_MOVE, PLANNING, NRML_MOVE, RIGHT_END_ON, and LEFT_END_ON.

Two data structures, property list and space-managing list, describe each situation of blocks world. The property list uses the frame concept to represent the relation between squares and a square and table, the location and size of a square, and a square's name. The space-managing list keeps track of where on the table there may be any empty room.

Finally, operations and results are illustrated including particular problems of a natural world.
CHAPTER V

CONCLUSION AND FURTHER WORKS

Conclusion

The main focus of this research is on implementing blocks world visually and realistically.

It is significant that a graphics system is used to visualize blocks world on a screen. Graphics data provides more information than character data. A person can understand each situation easily and quickly by using a graphics screen.

Realistic results in this study brought a pseudo world closer to a real world. The reality is an important factor to AI research because results of AI studies should contribute to not only virtual worlds in laboratories but also to real daily life.

The above two purposes were implemented in this research. In addition, both factors of this study have not been implemented in previous studies. Therefore, this study contributes to the study of blocks world as an example of AI research.

Further Works

Some further work is recommended from the view points of visualization and reality.

Visualization was implemented in this thesis. However, it is restricted by the two dimensional world. It is meaningful that
blocks world is constructed in the three dimensions. The issues which arise from constructing the three dimensional world might be related to other fields of AI: vision, pattern recognition, and so forth.

Although this thesis implemented a realistic aspect, there are still a number of other situations of a real world which have been ignored. For example, what happens if all squares do not have the same size? For example, when a large square rests on a small one and another small square is placed on a side of a large one (see Figure 23 (i)), in a real world the squares fall down.

![Figure 23](image)

**Figure 23.** Examples of Realistic Situations

Even if all squares have the same size, when each square rests on another one, locating the center of gravity of the above one on the edge of the below one (see Figure 23 (ii)), squares usually fall down. In theory, there exists a special situation in which squares do not fall down; however, in a real world squares fall down in almost all cases. These two situations are interesting problems and recommended for further research.
On the other hand, it is another interesting situation that in blocks world there are not only squares but also circles or triangles. Though blocks world in SHRDLU (Winograd, 1972) has not only blocks but also pyramids, it does not have spheres. It is a clear fact that any object cannot rest on triangles or circles. A program should consider this fact in blocks world and warn the user when this is attempted. It is difficult to implement those states including triangles and circles; however, those situation are interesting and realistic.

The above problems are not solved in this research, but could be interesting avenues of exploration.
APPENDICES
Appendix A

Hardware and Software Environment
Hardware and Software Environment

IBM-PC  IBM-PC XT with 10MB Hard Disk
        Main Memory 640K Byte
        Extended Graphics Adaptor

PC-DOS  Version 3.1
AutoCAD Version 2.52

Mouse  MicroSoft Mouse

Plotter  Graphtec MP1000

Plotter, IBM-PC, and mouse (left to right)
IBM-PC and mouse (left to right)

Plotter
User Manual

A user should be familiar with PC-DOS and AutoCAD.

Before Starting Program

1) Set Initial Values

In PC-DOS, the user should set the initial values, lispheap=40000 and lispstack=5000. If the user forgets to set these values, the error "insufficient node space" occurs.

2) AutoCAD Directory

If the AutoCAD system is in a subdirectory, the user should change the directory to the AutoCAD subdirectory.

Starting AutoCAD

1) Starting AutoCAD

The user starts the AutoCAD system. After getting the root menu screen of AutoCAD, the user must type "2", then press ENTER. Then type "block" and press ENTER. After finishing these operations, wait for several minutes until prompt "COMMAND: " appears.

Blocks World

1) Starting "Blocks World"

Type "(load "block")" followed by ENTER, and wait for a few minutes. After finishing it, type "(start)" followed by ENTER. Then the system starts. The system always displays what the user should type. At this point, users do as they please by following the instruction.

2) Create command

The create command is used to create a square. When a user types "create" or "CREATE," the system answers "Please give me the name of square." The system accepts any name, for example, even a name of only one character, or even a string which does not begin with letter.

If a square of the same name already exists, the system warns the user by the message, "That name already exists. Please consider another name."

The system needs several seconds to draw a square. A user should wait for the message, "create, move, or exit?"
If there is no room to create a square, the system displays the message, "You cannot create a square because of filling the table." Then the system asks "move, or exit?", and does not ask "create?"

3) Move command

The move command is used to move a square from one location to another by using a mouse.

When the user types "move" or "MOVE," the system answers "Please pick up the square which you want to move." After the user indicates any point inside the square, the system displays "Please indicate the point where you want to move the square." If the point which the user indicates is not inside any square, the system warns "I'm sorry. I cannot find the square which you have indicated."

4) Exit command

The exit command is used to exit the program. When a user types "exit" or "EXIT," the program terminates and returns back to AutoCAD. Drawing data is kept.

5) Error

When the user types another string which is not mentioned above, the system answers "I do not understand what you want to do. Please type exactly again." Then a message "create, move or exit?" appears on the screen.
Appendix C

Results
Results

In the following pictures, each square's label was typed by the author, because the pictures were not clear for duplication.

A sequence of commands is given below each picture. This sequence shows how each configuration was obtained. In the sequence of commands, normal character fonts are responses from the system, underlined characters are the user responses, and italics are operations using the mouse.
Picture 1. After creating a square using "create" command

Sequence of commands: create, move, or exit?

create
Please give the name of the square.
A
Picture 2. After creating three squares using "create" command

Sequence of commands: create, move, or exit?

create
Please give the name of the square.
A
create, move, or exit?
create
Please give the name of the square.
B
create, move, or exit?
create
Please give the name of the square.
C
Picture 3. After moving a square on another using "move" command
This result assumes that initial state is Picture 2.

Sequence of commands: create, move, or exit?

move
Please pick up the square which you want to move.
pick up the square "C"
Please indicate the point where you want to move the square.
indicate the point on the square "A"
After moving a square using "move" command
This result assumes that initial state is Picture 2.

Sequence of commands: create, move, or exit?

move
Please pick up the square which you want to move.

pick up the square "C"
Please indicate the point where you want to move the square.

indicate the point outside of the square "A"
Picture 5. After repeating "create" and "move"

Sequence of commands: create, move, or exit?

create
Please give the name of the square.
A
create, move, or exit?
create
Please give the name of the square.
B
create, move, or exit?
create
Please give the name of the square.
C
create, move, or exit?
move
Please pick up the square which you want to move.
pick up the square "B"
Please indicate the point where you want to move the square.
indicate the point on the square "A"

Repeat above mentioned sequence until obtaining Picture 5.
A process to move a square below another to another
A process before getting Picture 8.
This process assumes that initial state is Picture 5.

Sequence of commands: create, move, or exit?
move
Please pick up the square which you want to move.
pick up the square "B"
Please indicate the point where you want to move the square.
indicate the point on the square "E"
Picture 7. A process to move a square below another to another
A process before getting Picture 8.
This drawing appears the next to Picture 6.
The system draws this screen. Therefore, The user does
not need to type any data.
Picture 8. The result after moving a square onto another one
This result assumes that initial state is Picture 5.
This drawing appears the next to Picture 7.
The system draws this screen. Therefore, The user does
not need to type any data.
It takes a while after the user types a sequence of
commands below Picture 6.
Sequence of commands: create, move, or exit?

- **create**
  - Please give the name of the square.
  - A

- **create**, move, or exit?
  - **create**
  - Please give the name of the square.
  - B

- **create**, move, or exit?
  - **create**
  - Please give the name of the square.
  - C

- **move**
  - Please pick up the square which you want to move.
  - **pick up the square** "B"
  - Please indicate the point where you want to move the square.
  - **indicate the point on the square** "A"

Repeat above mentioned sequence until obtaining Picture 9.
Sequence of commands: create, move, or exit?
move
  Please pick up the square which you want to move.
  *pick up the square "B"*
  Please indicate the point where you want to move the square.
  *indicate the point outside of the square "E"*
Picture 11. After repeating "create" and "move"

Sequence of commands: create, move, or exit?

create
Please give the name of the square.
A
create, move, or exit?
create
Please give the name of the square.
B
create, move, or exit?
create
Please give the name of the square.
C
create, move, or exit?
move
Please pick up the square which you want to move.
pick up the square "B"
Please indicate the point where you want to move the square.
indicate the point on the square "A"

Repeat above mentioned sequence until obtaining Picture 11.
Picture 12. A process to move a square to another one
A process before getting Picture 14.
This process assumes that initial state is Picture 11.

Sequence of commands: create, move, or exit?
- move
  Please pick up the square which you want to move.
  *pick up the square "B"
  Please indicate the point where you want to move the square.
  *indicate the point on the square "G"
Picture 13. A process to move a square to another one
A process before getting Picture 14.
This drawing appears after a while of getting
Picture 12.
The system draws this screen. Therefore, The user does
not need to type any data.
Picture 14. Result after moving a square onto another one
This result assumes that initial state is Picture 11.
This drawing appears the next to Picture 13.
The system draws this screen. Therefore, The user does not need to type any data.
It takes a while after the user types a sequence of commands below Picture 12.
Sequence of commands: create, move, or exit?

create
Please give the name of the square.
A
create, move, or exit?
create
Please give the name of the square.
B
create, move, or exit?
create
Please give the name of the square.
C
create, move, or exit?
move
Please pick up the square which you want to move.
pick up the square "B"
Please indicate the point where you want to move the square.
indicate the point on the square "A"

Repeat above mentioned sequence until obtaining Picture 15.
Picture 16. A process to move a square to another one
A process before getting Picture 18.

Sequence of commands: create, move, or exit?

**move**
Please pick up the square which you want to move.
*pick up the square "B"*
Please indicate the point where you want to move the square.
*indicate the point outside of the square "F"*
Picture 17. Result after moving a square
This result assumes that initial state is Picture 15.
This drawing appears after a while of getting Picture 16.
The system draws this screen. Therefore, The user does not need to type any data.
It takes a while after the user types a sequence of commands below Picture 15.
Appendix D

Source Program
When you want to create a square, type create.

Please wait for a few seconds.

create, move, or exit?

;**********************************************************************************************

; Initial

;**********************************************************************************************

(defun initial ()
  (setvar "cmdecho" 0) ; Set echo mode
  (setq dflt_width 2.0) ; Default width of square
  (setq dflt_half_width 1.0) ; dflt_width/2
  (setq dflt_height 1.0) ; Default height of square
  (setq dflt_half_height 0.5) ; dflt_height/2
  (setq table 'empty) ; Set empty to table
  (setq square_list '()) ; Set nil list
  (setq space_list '((12.0 (1.0 13.0))) ; Set x lower limit
  (setq x_lower_limit 1.0)
  (setq x_upper_limit 13.0)
  (setq y_lower_limit 2.0)
  (setq y_upper_limit 9.0)
  (command "textscr")
  (print "******************************************************************************************")
  (print)
  (prompt "When you want to create a square, ")
  (prompt "type create.")
  (print "******************************************************************************************")
  (print)
  (print)
  (prompt "Please wait for a few seconds.")
  (print)
  (print)
  (command "delay" 3000)
  (command "graphscr")
  (print)
  (prompt "create, move, or exit?")
  ) ; (defun initial )

(defun start ()
  (initial)
  (idle)
  ) ; (defun start
(DEFUN AGAIN ()
  (PRINT)
  (PROMPT "create, move, or exit?")
  (IDLE)
) ; (DEFUN AGAIN

(DEFUN IDLE ()
  (PRINT)
  (SETQ input (GETSTRING))
  (WHILE (NOT (OR (EQUAL input ""e")
                   (EQUAL input "EXIT")
                   (EQUAL input "e")
                   (EQUAL input "E")
                   )
    ; (OR
    ) ; (NOT
    (COND ( ; case of "create" <---------------------
      (OR (EQUAL input "create")
          (EQUAL input "CREATE")
          (EQUAL input "c")
          (EQUAL input "C")
          )
    ) ; OR
    (CREATE_SQUARE) ; Call CREATE_SQUARE
    ) ; end case of "create" <---------------------
    ; case of "move" <---------------------
    (OR (EQUAL input "move")
        (EQUAL input "MOVE")
        (EQUAL input "m")
        (EQUAL input "M")
        )
    ) ; (OR
    (MOVE_SQUARE) ; Call MOVE_SQUARE
    ) ; end case of "move" <---------------------
    ; the other case <---------------------
    T
    (PROGN
      (PRINT)
      (PROMPT "I'm sorry. ")
      (PROMPT "I do not understand what you ")
      (PROMPT "want to do. ")
      (PROMPT "Please type exactly again.")
      ) ; (PROGN
    ) ; end the other case <---------------------
) ; (COND

(IF (EQUAL table 'full)
(PROGN ; table full
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
  (PRINT)
'old_start 'old_end ; ;
'new1_end 'new1_list ; ;
'new2_start 'new2_list)) ; ;
(SETQ square_list ; ;
(CONS (LIST name ; ;
 (LIST ; ;
 (LIST 'location ; ;
 (LIST x_on_table ;
 y_on_table));
 ) ; (LIST 'location ; ;
 (LIST 'on 'table); ;
 (LIST 'size ; ;
 (LIST dflt_half_width ;
 dflt_half_height)
 ) ; (LIST 'size ; ;
 ) ; (LIST ; ;
 ) ; (LIST name ; ;
 square_list ; ;
 ) ; (CONS ; ;
 ) ; (SETQ square_list ; ;
 (SETQ pt_on_table ; ;
 (LIST x_on_table y_on_table); ;
 ) ; (SETQ pt_on_table ; ;
 (COMMAND "INSERT" "SQUARE" ; ;
 pt_on_table 1 1 0 name); ;
 (CLEAN (LIST 'name 'pt_on_table)); ;
 ) ; (SETQ ; ;
 (PROGN space exists <--------- ;
 space full <--------- ;
 (SETQ table 'full); ;
 (PRINT); ;
 (PROMPT "You can not create "); ;
 (PROMPT "a square because of "); ;
 (PROMPT "filling the table."); ;
 ) ; (PROGN <---------- ;
 ) ; (IF (CHECK_TABLE) ; ;
 ) ; (PROGN name is new <---------- ;
 ) ; (IF (ASSOC name square_list) ; ;
 ) ; (PROGN table empty <----------
 ) ; (DEFUN CREATE_SQUARE
(DEFUN CHECK_TABLE ()
  (SETQ m 0)
  (WHILE (AND (< m (LENGTH space_list)) ; <--------------
                (> dflt_width (CAR (NTH m space_list))) ;
                ) ;(>)
    ) ;(AND
    (SETQ m (1+ m))
  ) ;(WHILE ; ; Check table *
    (IF (< m (LENGTH space_list))
      (PROGN ;table has any space <--------------
              (SETQ old_list (NTH m space_list))
              (SETQ old_start (CAADR old_list))
              (SETQ old_end (LAST (CADR old_list))
              ) ;(SETQ
              (SETQ x_on_table (+ old_start
                                (/ (- old_end old_start)
                                   2
                                )
                                ) ;(/ ;(SETQ
                                (SETQ new1_end (- x_on_table dflt_half_width))
                                (SETQ new1_list (LIST old_start new1_end))
                                (SETQ new1_list (CONS (- new1_end old_start)
                                                  (CONS new1_list '())))
                                ) ;(CONS
                                (SETQ new2_start (+ x_on_table dflt_half_width))
                                (SETQ new2_list (LIST new2_start old_end))
                                (SETQ new2_list (CONS (- old_end new2_start)
                                                  (CONS new2_list '())))
                                ) ;(CONS
                                (SETQ space_list (REPLACE_1to2 space_list
                                                   old_list
                                                   (LIST new1_list
                                                        new2_list)
                                                   ) ;(REPLACE_1to2
                             ) ;(IF table has any space <--------------
      ) ;(IF (< m (LENGTH space_list))
  ) ;(DEFUN CHECK_TABLE
(DEFUN REPLACE_lto2 (list old_element new_element)
  (SETQ first_half (REVERSE (CDR (MEMBER old_element list))
   (REVERSE list)
   (MEMBER old_element list)
   (CDR)
   (SETQ second_half (CDR (MEMBER old_element list))
   (CDR)
   (APPEND first_half new_element second_half)
   (DEFUN REPLACE_lto2

; Move square

;***********************************************************
; Replace element of list 1 to 2
;***********************************************************

;***********************************************************

(DEFUN MOVE_SQUARE ()
  (PRINT)
  (PROMPT "Please pick up the square ")
  (PRINT)
  (PROMPT "which you want to move. ")
  (PRINT)
  (SETQ point (GETPOINT))
  (SETQ point_x (CAR point))
  (SETQ point_y (CADR point))
  (IF (CHK_BFR_MOVE square_list point_x point_y)
    (PROGN ;can move <---------------------
      (PRINT)
      (PROMPT "Please indicate the point ")
      (PROMPT "where you want to move the ")
      (PROMPT "square.")
      (SETQ point (GETPOINT))
      (CHK_AFT_MOVE square_list moving_square point)
    ) ;(PROGN end can move <---------------------
    (PROGN ;can not move <---------------------
      (PRINT)
      (PROMPT "I'm sorry. ")
      (PROMPT "I can not find the square which you ")
      (PROMPT "have indicated.")
    ) ;(PROGN end can not move <---------------------
  ) ;(IF (CHK_BFR_MOVE
  (CLEAN (LIST 'point 'point_x 'point_y
    'trgt_loc 'trgt_size
    'x_trgt_loc 'y_trgt_loc
    'trgt_half_width 'trgt_half_height))

  ) ;(DEFUN MOVE_SQUARE
;;;***********************************************************************************
;; * Check before move *
;; *
;;***********************************************************************************
(DEFUN CHK_BFR_MOVE (property_list point_x point_y)
  (setq n 0)
  (SRCH_LOC_SIZE (CADR (NTH n property_list)))
  (while ;;-----------------------------<---------------
    (not (and (<= (- x_trgt_loc trgt_half_width) point_x)
                (+ x_trgt_loc trgt_half_width) ; «=
                (<= (- y_trgt_loc trgt_half_height) point_y)
                (+ y_trgt_loc trgt_half_height) ; «=)
      ; (AND
      (setq n (1+ n))
      (SRCH_LOC_SIZE (CADR (NTH n property_list)))
    ) ; (WHILE ;-------------------------<--------------
  ) ; (IF (< n (LENGTH property_list))
  ) ; (DEFUN CHK_BFR_MOVE
;;***********************************************************************************
;; Search location & size *
;; *
;;***********************************************************************************
(DEFUN SRCH_LOC_SIZE (target_list)
  (if target_list
    (progn ;-------------------------<--------------
      (setq trgt_loc (cadr (assoc 'location target_list)))
      (setq x_trgt_loc (car trgt_loc))
      (setq y_trgt_loc (cadr trgt_loc))
      (setq trgt_size (cadr (assoc 'size target_list)))
      (setq trgt_half_width (car trgt_size))
      (setq trgt_half_height (cadr trgt_size))
    ) ; (PROGN
    (progn
      (setq x_trgt_loc (/ (- x_upper_limit x_lower_limit) 2))
      (setq trgt_half_width (1+ x_trgt_loc x_lower_limit))
      (setq trgt_half_height (1+ y_trgt_loc y_upper_limit))
    ) ; (PROGN
  ) ; (IF
) ; (DEFUN SRCH_LOC_SIZE

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(DEFUN MOVE_DRAW (object_property new_loc)
  (SRCH_LOC_SIZE object_property)
  (SETQ left_lower (LIST (- x_trgt_loc trgt_half_width)
    (- y_trgt_loc trgt_half_height))
    )
  (SETQ right_upper (LIST (+ x_trgt_loc trgt_half_width)
    (+ y_trgt_loc trgt_half_height))
    )
  (COMMAND "MOVE" "W" left_lower right_upper ""
    trgt_loc new_loc)
  (COMMAND "REDRAW")
  (CLEAN (LIST 'trgt_loc 'trgt_size 'x_trgt_loc 'y_trgt_loc
    'trgt_half_width 'trgt_half_height
    'left_lower 'right_upper))
  )
)

(DEFUN CHK_AFT_MOVE (object_list moving_object location)
  (SETQ new_x (CAR location))
  (SETQ new_y (CADR location))
  (SETQ moving_property (CADR moving_object))
  (SETQ moving_size (CAR (ASSOC 'size moving_property)))
  (SETQ moving_half_width (CAR moving_size))
  (SETQ moving_half_height (CADR moving_size))
  (SETQ n 0)
  (SRCH_LOC_SIZE (CADR (NTH n object_list)))
  (WHILE (OR (< (+ new_x moving_half_width)
    (- x_trgt_loc trgt_half_width))
    (< (+ x_trgt_loc trgt_half_width)
    (- new_x moving_half_width))
    )
    )
  (OR
    (SETQ n (1+ n))
    (SRCH_LOC_SIZE (CADR (NTH n object_list))))
  )
  (WHILE)
  (IF (< n (LENGTH object_list))
    (PROGN
      (IF (ASSOC 'below moving_property)
        (PLANNING moving_object))
      (SRCH_LOC_SIZE (CADR (NTH n object_list)))
      (COND ( ;case of normal

----------
(<= (- x_trgt_loc trgt_half_width) new_x
(<= (+ x_trgt_loc trgt_half_width) new_x
)} ;(<=
(NRML_MOVE new_x new_y
 object_list
 (NTH n object_list)
 moving_object)
) ;end case of normal

(<= (- x_trgt_loc trgt_half_width)
 (- new_x moving_half_width)
 (+ x_trgt_loc trgt_half_width)
)} ;(<=
(LEFT_END_ON new_x new_y
 (NTH n object_list)
 moving_property)
) ;end case of left end on the square

(<= (- x_trgt_loc trgt_half_width)
 (+ new_x moving_half_width)
 (+ x_trgt_loc trgt_half_width)
)} ;(<=
(RIGHT_END_ON new_x new_y
 (NTH n object_list)
 moving_property)
) ;end case of right end on the square

; end case of on table to on table

;not on the other object
(COND (;over x limits
 (OR (<= x_upper_limit new_x)
 (<= new_x x_lower_limit))
 ) ;OR
 (PROGN
 (PRINT)
 (PROMPT "Your square will drop out from ")
 (PROMPT "the table. I can not place it ")
 (PROMPT "on that point.")
 ) ;(PROGN
) ;end over x limits

;over y limits
(OR (<= y_upper_limit new_y)
 (<= new_y y_lower_limit))
 ) ;(OR
 (PROGN
 (PRINT)
 (PROMPT "I'm sorry. I can not place it ")
 (PROMPT "on that point.")
 ) ;(PROGN
) ;end over y limits

;on table to on table
(PROGN
  (IF (ASSOC 'below moving_property)
    (PLANNING moving_object))
  (SETQ x_after_move new_x)
  (SETQ y_after_move y_on_table)
  (MDFY_SPACE_LIST moving_property)
  (MOVE_DRAW moving_property)
  (LIST x_after_move y_after_move)
  (MOVE_DRAW)
  (MDFY_LOCATION moving_object)
  (MDFY_RELATIONSHIP_ON moving_object 'table)
  (MDFY_RELATIONSHIP_BELOW moving_object 'table))
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
(SETQ y_of_check (CADR check_loc))
(SETQ check_size (CADR (ASSOC 'size object_property))
 ) ; (CADR

(SETQ check_half_width (CAR check_size))
(SETQ check_half_height (CADR check_size))
(IF (<= (+ x_of_check check_half_width) new_x
 (+ x_of_check check_half_width)
); (<=
 (PROGN ; new_x is on the other square<------------
 (IF (<= (+ y_of_check check_half_height) new_y)
  ; (<=
 (PROGN ; normal move <--------
 (SETQ x_after_move new_x)
 (SETQ y_after_move (+ y_of_check
  ;
  (* check_half_height 2));
  ) ; (+
  ;
  (PROGN ;)
  ;)

  ;
  (MDFY_SPACE_LIST moving_property)

  ;
  (MOVE_DRAW moving_property)

  ;
  (LIST x_after_move y_after_move);

  ) ; (MOVE_DRAW
  ;)

  ;
  (MDFY_LOCATION moving_object)

  ;
  (MDFY_RELATIONSHIP_ON moving_object
  ;
  (CAR object));

  ;
  (MDFY_RELATIONSHIP_BELOW moving_object
  ;
  (CAR object));

  ) ; (PROGN end normal move <--------------
  ;
  (PRINT)

  ;
  (PROMPT "Your indication is in the ")

  ;
  (PROMPT "other square. I can not move ")

  ;
  (PROMPT "it to that point."
  ) ; (PROGN end new_y within the square<-----
  ) ; (IF (<= (+ y_of_check check_half_height) new_y)
  ) ; (PROGN end new_x within the square<-----
  ) ; (IF (<= (+ y_of_check check_half_height) new_y)
  ) ; (PROGN end new_x on the other square<--------
  ) ; (PROGN ; new_x not on the other square<---------
  ) ; (PRINT)

  ;
  (PROMPT "Your operation does not have any ")

  ;
  (PROMPT "mistakes.")

  ;
  (PRINT)

  ;
  (PROMPT "However, I'm sorry, I can not place ")

  ;
  (PROMPT "the square at that point."
  ) ; (PROGN end new_x on the other square<--------

  ) ; (IF (<=
  (CLEAN (LIST 'check_loc 'x_of_check 'y_of_check
  'check_size
  'check_half_width 'check_half_height
  'new_x 'new_y 'x_after_move 'y_after_move))
  ) ; (DEFUN NRML_MOVE


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(DEFUN MDFY_SPACE_LIST (moving_object)
  (IF (EQUAL 'table (CADR (ASSOC 'on moving_object)))
    (IF (= y_on_table y_after_move)
        (CHNG_SPACE_LIST moving_object)
        (ENLRG_SPACE_LIST moving_object)))
  (IF (= y_on_table y_after_move)
    (SHRNK_SPACE_LIST moving_object)))
)

(DEFUN ENLRG_SPACE_L 1ST (moving_object)
  (SETQ x_location (CAADR (ASSOC 'location moving_object)))
  (SETQ x_left_end (- (CAADR (ASSOC 'location moving_object))
                      dflt_half_width))
  (SETQ m 0)
  (WHILE (AND (< m (LENGTH space_list))
              (NOT (EQ (LAST (CADR (NTH m space_list)))
                       x_left_end))
    (SETQ m (1+ m)))
  (IF (< m (LENGTH space_list))
    (PROGN
      (SETQ old_list (NTH m space_list))
      (SETQ new_start (CAADR old_list))
      (SETQ old_list (NTH (1+ m) space_list))
      (SETQ new_end (CADADR old_list))
      (SETQ new_size (- new_end new_start))
      (SETQ new_list (LIST new_start new_end))
      (SETQ new_list (CONS new_size (CONS new_list '()))
           (CONS space_list (REPLACE_2tol space_list m
                               (LIST new_list)))
    )
  )
(CLEAN (LIST 'old_list 'new_list 'new_start 'new_end 'new_size))

(SETQ table 'empty)
)

(IF (< m (LENGTH space_list))
 (CLEAN (LIST 'x_left_end))
)

; (DEFUN ENLRG_SPACE_LIST
 ;*************************
 ;* Shrink space list after move *
 ;*************************
 ;
 (DEFUN SHRNK_SPACE_LIST (moving_object)
 
 (SETQ x_left_end (- x_after_move dflt_half_width))
 (SETQ x_right_end (+ x_after_move dflt_half_width))
 (SETQ m 0)
 (WHILE (AND (< m (LENGTH space_list))
 
 
 (<= (CAADR (NTH m space_list))
 
 
 x_left_end)
 
 
 (<= x_right_end
 
 
 (LAST (CADR (NTH m space_list))
 
 
 ) ;(LAST
 
 
 ) ;(<=
 
 
 ) ;(AND
 
 
 (SETQ m (1+ m))
 
 ) ;(WHILE
 
 (IF (< m (LENGTH space_list))
 
 
 (PROGN
 
 
 (SETQ old_list (NTH m space_list))
 (SETQ old_start (CAADR old_list))
 (SETQ old_end (LAST (CADR old_list))))
 (SETQ newl_end x_left_end)
 (SETQ newl_list (LIST old_start newl_end))
 (SETQ newl_list (CONS (- newl_end old_start)
 
 
 (CONS newl_list '()))
 ) ;(CONS
 
 
 (SETQ new2_start x_right_end)
 (SETQ new2_list (LIST new2_start old_end))
 (SETQ new2_list (CONS (- old_end new2_start)
 
 
 (CONS new2_list '()))
 ) ;(CONS
 
 
 (SETQ space_list (REPLACE_lto2 space_list
 
 
 old_list
 
 
 (LIST newl_list
 
 
 new2_list)
 
 
 ) ;(REPLACE
 
 
 ) ;(SETQ
 
 
 ) ;(PROGN
 
 
 ) ;(IF
 
 
 ) ;(DEFUN SHRNK_SPACE_LIST

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(DEFUN CHNG_SPACE_LIST (moving_object)
  (SETQ x_left_end (- (CAADR (ASSOC 'location moving_object))
                      dflt_half_width))
  (SETQ m 0)
  (WHILE (AND (< m (LENGTH space_list))
               (NOT (EQ (LAST (CADR (NTH m space_list)))
                        x_left_end)))
    (SETQ m (1+ m)))
  (IF (< m (LENGTH space_list))
    (PROGN
      (SETQ old_list (NTH m space_list))
      (SETQ old_start (CAADR old_list))
      (SETQ new_end (- x_after_move dflt_half_width))
      (SETQ new_size (- new_end old_start))
      (SETQ new_list (CONS new_size
                      (CONS (LIST old_start new_end)
                            '())))
      (SETQ space_list (SUBST new_list old_list space_list))
      )
    )
  (SETQ x_right_end (+ (CAADR (ASSOC 'location moving_object))
                       dflt_half_width))
  (SETQ m 0)
  (WHILE (AND (< m (LENGTH space_list))
               (NOT (EQ (CADR (NTH m space_list))
                        x_right_end)))
    (SETQ m (1+ m)))
  (IF (< m (LENGTH space_list))
    (PROGN
      (SETQ old_list (NTH m space_list))
      (SETQ old_end (LAST (CADR old_list)))
      (SETQ new_start (+ x_after_move dflt_half_width))
      (SETQ new_size (- old_end new_start))
      (SETQ new_list (CONS new_size
                      (CONS (LIST old_end new_start)
                            '())))
      (SETQ space_list (SUBST new_list old_list space_list))
      )
    )
  )
)
(CONS (LIST new_start old_end) '())

) ;(CONS
  (SETQ space_list (SUBST new_list old_list space_list))
) ;(PROGN
  (IF
    (CLEAN (LIST 'x_right_end 'old_list 'old_end 'new_start 'new_size 'new_list))
  ) ;(DEFUN CHNG_SPACE_LIST
    ;*******************************************************************
    ; Replace element of list 2 to 1
    ;*******************************************************************
    (DEFUN REPLACE_2to1 (list m new_element)
      (SETQ old_element_1 (NTH m list))
      (SETQ first_half (REVERSE (CDR (MEMBER old_element_1 (REVERSE list))
      ) ;(MEMBER
        ) ;(REVERSE
      ) ;(SETQ
      (SETQ old_element_2 (NTH (1+ m) list))
      (SETQ second_half (CDR (MEMBER old_element_2 list))
      ) ;(SETQ
      (APPEND first_half new_element second_half)
      ) ;(DEFUN REPLACE_2TO1
    ;*******************************************************************
    ; Modify location of property list after move
    ;*******************************************************************
    (DEFUN MDFY_LOCATION (object)
      (SETQ object_name (CAR object))
      (SETQ old_list (CADR (ASSOC object_name square_list))
      ) ;(SETQ
      (SETQ old_location (ASSOC 'location old_list))
      (SETQ new_location (LIST 'location
          (LIST x_after_move y_after_move))
      ) ;(SETQ
      (SETQ new_list (CON new_location old_list)
      (SETQ old_list (ASSOC object_name square_list))
      (SETQ new_list (CONS object_name (CONS new_list '())))
      ) ;(SETQ
      (SETQ square_list (SUBST new_list old_list square_list))
      (CLEAN (LIST 'object_name 'old_list 'new_list 'old_location 'new_location))
    ) ;(DEFUN MDFY_LOCATION
(DEFUN MDFY_RELATIONSHIP_ON (object supporting_object_name)
  (SETQ object_name (CAR object))
  (SETQ old_list (CADR (ASSOC object_name square_list)))
  (SETQ old_relationship (ASSOC 'on old_list))
  (SETQ new_relationship (LIST 'on supporting_object_name))
  (SETQ new_list (SUBST new_relationship old_relationship old_list))
  (SETQ old_list (ASSOC object_name square_list))
  (SETQ new_list (CONS object_name (CONS new_list '())))
  (SETQ square_list (SUBST new_list old_list square_list))
  (CLEAN (LIST 'object_name 'old_list 'new_list 'old_relationship 'new_relationship)))

(DEFUN MDFY_RELATIONSHIP_BELOW
  (object supporting_object_name)
  (SETQ object_name (CAR object))
  (SETQ m 0)
  (WHILE (NTH m square_list)
    (IF (EQUAL object_name
        (CADR (ASSOC 'below
            (CADR (NTH m square_list))
        ) ;(ASSOC
      ) ;(EQUAL
        (PROGN
          (SETQ old_list (CADR (NTH m square_list)))
        ) ;(SETQ
          (SETQ old_relationship (ASSOC 'below old_list))
          (SETQ new_list (DELETE old_list old_relationship))
        )
      (SETQ old_list (NTH m square_list))
      (SETQ new_list (CONS (CAR (NTH m square_list))
        (CONS new_list '()))
    ) ;(CONS
  ) ;(SETQ
  (SETQ square_list (SUBST new_list old_list square_list))
  (CLEAN (LIST 'old_list 'new_list 'object_name 'old_list 'new_list 'old_relationship 'new_relationship)))
'old_relationship)')

) ; (PROGN
) ; (IF
(SETQ m (+ m))
) ; (WHILE
(IF (NOT (EQ supporting_object_name 'table))
(SETQ old_list (CDR (ASSOC supporting_object_name square_list)))
) ; (SETQ
(SETQ new_relationship (LIST 'below object_name))
(SETQ new_list (APPEND old_list
(CONS new_relationship '())))
) ; (SETQ
(SETQ old_list (ASSOC supporting_object_name square_list))
(SETQ new_list (CONS supporting_object_name
(CONS new_list '())))
) ; (SETQ
(SETQ square_list (SUBST new_list old_list square_list))
(CLEAN (LIST 'object_name 'old_list 'new_list
'new_relationship))
)
) ; (PROGN
) ; (IF (NOT (EQ
) ; (DEFUN MDFY_RELATIONSHIP_BELOW
; Delete element from list
; 
;******************************************************************************
(DEFUN DELETE (list element)
(SETQ first_half (REVERSE (CDR (MEMBER element
(REVERSE list)
) ; (MEMBER
) ; (CDR
) ; (REVERSE
) ; (SETQ
(SETQ second_half (CDR (MEMBER element list)))
) ; (SETQ
(APPEND first_half second_half)
) ; (DEFUN DELETE

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(DEFUN LEFT_END_ON (new_x new_y object move_property)
  (MOVE_BFR_DECLINE new_x new_y (CADR object) move_property)
  (SETQ x_center_of_rotation (+ x_of_check check_half_width))
  (SETQ y_center_of_rotation (+ y_of_check check_half_height))
  (SETQ center_of_rotation (LIST x_center_of_rotation y_center_of_rotation))
  (SRCH_LOC_SIZE move_property)
  (SETQ move_half_width trggt_half_width)
  (SETQ move_half_height trggt_half_height)
  (SETQ distance (ABS (- (+ x_after_move move_half_width)
                     x_center_of_rotation))
  (SETQ x_contact_point (+ x_center_of_rotation
                      (SQRT (- (* distance distance)
                         (* 2 check_half_height)
                         (* 2 check_half_height)
                      )
  (SETQ y_contact_point (- y_of_check check_half_height)
  (SETQ contact_point (LIST x_contact_point y_contact_point))
  (SETQ decline_radians (ANGLE center_of_rotation contact_point))
  (SETQ rotation_angle
       (- (ATOF (ANGTOS decline_radians 0 2))
          360.0))
  (DECLINE_DRAW x_after_move y_after_move
                move_half_width move_half_height
                center_of_rotation rotation_angle)
  (CLEAN (LIST 'new_x 'new_y
                'x_center_of_rotation 'y_center_of_rotation
                'center_of_rotation 'rotation_angle
                'move_half_width 'move_half_height
                'distance 'decline_radians)
(DEFUN RIGHT_END_ON (new_x new_y object move_property)
  (MOVE_BFR_DECLINE new_x new_y (CADR object) move_property)
  (SETQ x_center_of_rotation (- x_of_check check_half_width))
  (SETQ y_center_of_rotation (+ y_of_check check_half_height))
  (SETQ center_of_rotation (LIST x_center_of_rotation y_center_of_rotation))
  (SRCH_LOC_SIZE move_property)
  (SETQ move_half_width trg_ht_half_width)
  (SETQ move_half_height trg_ht_half_height)
  (SETQ distance (ABS (- x_center_of_rotation (- x_after_move move_half_width)) ;)
    (SETQ x_contact_point (- x_center_of_rotation
      (SQRT (- (* distance distance) (* 2 check_half_height) (* 2 check_half_height)) ;)
        (SETQ y_contact_point (- y_of_check check_half_height)
      ) ;
    ) ;
  ) ;
  (SETQ contact_point (LIST x_contact_point y_contact_point))
  (SETQ decline_radians (ANGLE center_of_rotation contact_point))
  (SETQ rotation_angle
    (- (ATOF (ANGTOS decline_radians 0 2)) 180.0)
  ) ;
  (DECLINE_DRAW x_after_move y_after_move
    move_half_width move_half_height center_of_rotation rotation_angle)
  (CLEAN (LIST 'new_x 'new_y
    'x_center_of_rotation 'y_center_of_rotation
  )

(DEFUN RIGHT_END_ON
  ; Move before declining
  ;
  ;*****************************
  (DEFUN MOVE_BFR_DECLINE (new_x new_y object_property move_property)
    (SRCH_LOC_SIZE object_property)
    (SETQ x_of_check x_trgt_loc)
    (SETQ y_of_check y_trgt_loc)
    (SETQ check_half_width trgt_half_width)
    (SETQ check_half_height trgt_half_height)
    (CLEAN (LIST 'x_trgt_loc 'y_trgt_loc
                 'trgt_half_width 'trgt_half_height))
    (SETQ x_after_move new_x)
    (SETQ y_after_move (+ y_of_check (* check_half_height 2)))
    (MOVE_DRAW move_property (LIST x_after_move y_after_move))
  )

;;*****************************
(DEFUN DECLINE_DRAW (x_after_move y_after_move move_half_width move_half_height
  center_of_rotation rotation_angle)
  (SETQ left_lower (LIST (- x_after_move move_half_width)
                          (- y_after_move move_half_height))
   (LIST)
  (SETQ right_upper (LIST (+ x_after_move move_half_width)
                         (+ y_after_move move_half_height))
   (LIST)
  (COMMAND "ROTATE" "W" left_lower right_upper ""
            center_of_rotation rotation_angle)
  (COMMAND "REDRAW")
  (CLEAN (LIST 'left_lower 'right_upper))
)
(DEFUN PLANNING (moving-object)
  (SETQ goal (CAR moving-object))
  (SETQ object-property (CADR moving-object))
  (WHILE (ASSOC 'below object-property)
    (SETQ object
      (ASSOC (CADR (ASSOC 'below
                    object-property))
        square-list))
    (SETQ object-property (CADR object)))
  ) ; (WHILE
  (IF (CHECK_TABLE)
    (PROGN
      (SETQ x-after-move x-on-table)
      (SETQ y-after-move y-on-table)
      (SETQ supporting-object-name 'table)
    ) ; (PROGN
    (PROGN
      (SRCH-LOC-SIZE object-property)
      (SETQ goal-sprt_loc_x x-trgt_loc)
      (SETQ goal-sprt-half-width trgt-half-width)
      (SETQ m 0)
      (SRCH-LOC-SIZE (CADR (NTH m square-list)))
      (WHILE (OR (<= (- new_x moving-half-width)
                       (+ x-trgt_loc trgt-half-width))
                (<= (- new_x moving-half-width)
                       (- x-trgt_loc trgt-half_width))
                (<= (- goal_sprt_loc_x
                        goal-sprt-half_width)
                        x-trgt_loc)
                (<= (+ goal_sprt_loc_x
                        goal-sprt-half-width))
        (SETQ m (+ m))
        (SRCH-LOC-SIZE (CADR (NTH m square-list)))
      ) ; (WHILE
      (SETQ temp-object (NTH m square-list))
      (SETQ object-property (CADR temp-object))
      (WHILE (ASSOC 'below object-property)
        (SETQ temp-object
          (ASSOC (CADR (ASSOC 'below
                          object-property))
            square-list))
        (SETQ object-property (CADR temp-object)))
      ) ; (WHILE
      (SRCH-LOC-SIZE object-property)
      (SETQ x-after-move x-trgt_loc)
    )) ; (IF
  (SETQ temp-object (NTH m square-list))
  (SETQ object-property (CADR temp-object))
  (WHILE (ASSOC 'below object-property)
    (SETQ temp-object
      (ASSOC (CADR (ASSOC 'below
                    object-property))
        square-list))
    (SETQ object-property (CADR temp-object)))
  ) ; (WHILE
  (SRCH-LOC-SIZE object-property)
  (SETQ x-after-move x-trgt_loc)
(SETQ y_after_move (+ y_trgt_loc
  (* 2 trgt_half_height)))
(SETQ supporting_object_name (CAR temp_object))
) ; (PROGN
) ; (IF
(CLEAN (LIST 'goal_sprt_loc_x 'goal_sprt_half_width
  'temp_object 'x_trgt_loc 'y_trgt_loc
  'trgt_half_width 'trgt_half_height))
(BFR_GOAL_MOVE x_after_move y_after_move object
  supporting_object_name)
) ; (DEFUN PLANNING
;***********************************************************************
; Before goal move
;***********************************************************************
(DEFUN BFR_GOAL_MOVE (x_after_move y_after_move object
  supporting_object_name)
(SETQ object_property (CADR object))
(WHILE (NOT (EQ goal (CAR object)))
(SETQ next_object_name (CADR
  (ASSOC 'on
    object_property)))
(IF (NOT (EQ supporting_object_name 'table))
  (MDFY_SPACE_LIST object_property))
  (MOVE_DRAW object_property
    (LIST x_after_move y_after_move))
  (MDFY_LOCATION object
    supporting_object_name)
  (MDFY_RELATIONSHIP_ON object
    supporting_object_name)
  (SETQ supporting_object_name (CAR object))
  (SETQ object (ASSOC next_object_name square_list))
  (SETQ object_property (CADR object))
  (SETQ y_after_move (+ y_after_move dflt_height))
) ; (WHILE
) ; (DEFUN BFR_GOAL_MOVE

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BIBLIOGRAPHY


Chiles, J. R. "Breaking Codes was This Couple's Lifetime Career." Smithsonian, June 1987, pp. 128-144.


