

California State University, San Bernardino

CSUSB ScholarWorks

Theses Digitization Project

John M. Pfau Library

2004

Develop heuristics to the popular Minesweeper game

Angela Tzujui Huang

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Software Engineering Commons](#)

Recommended Citation

Huang, Angela Tzujui, "Develop heuristics to the popular Minesweeper game" (2004). *Theses Digitization Project*. 2545.

<https://scholarworks.lib.csusb.edu/etd-project/2545>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

DEVELOP HEURISTICS TO THE POPULAR MINESWEEPER GAME

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Angela Tzujui Huang
September 2004

DEVELOP HEURISTICS TO THE POPULAR MINESWEEPER GAME

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by

Angela Tzujui Huang

September 2004

Approved by:



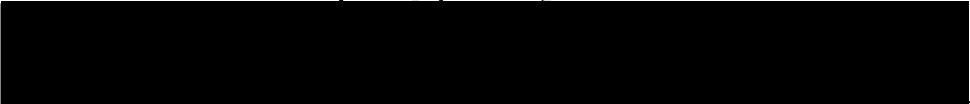
Dr. Richard Botting, Chair, Computer Science

Aug 25/04

Date



Dr. Kerstin Voigt, Computer Science



Dr. Ernesto Gomez, Computer Science

ABSTRACT

The mine sweeper game is a popular computer game, which can be viewed as a logic game or a probability game. There are certain cases where you can decide if there is or is not a mine. However, there are also cases that you can not decide whether there is a mine or not and you have to guess according to the probability. This project develops heuristics to help users solving the problem quickly with high success rates.

This project applies three colors: red, yellow, and green to covered squares to show the risk probability. The red color represents the highest risk square which has definitely a bomb or mine underneath, and the green color represents a safe square which has no bomb, and the yellow color represents an undecided square which may or may not have a bomb. The yellow color in a square has a size proportional to the risk. In the beginning, user should randomly guess several steps to get the uncovered area as large as possible to achieve the minimal time performance. Then, the heuristic will find safe uncovered squares as well as squares with bombs and assign green and red to them respectively. Besides, this project will calculate probability for each square which is neither red nor green. Furthermore, the probabilities of overlap conditions and

specific patterns, which are derived or obtained from simulation, are applied.

ACKNOWLEDGMENTS

Without the support and encouragement of my advisor, my family, and my friends, this project would not have been completed successfully. I am so grateful to Dr. Botting, who is such a wonderful and perfect advisor, for his kind advice, guidance, and suggestions. I also am grateful to Dr. Kerstin Voigt and Dr. Ernesto Gomez to be my committee and provide helpful comments on this project.

I appreciate my family for all their love, understanding and tolerance. My husband not only supports me in learning and living but also helps to take care of my son and daughter so I have enough time to finish my works. My sisters and brother always encourage me throughout my study in many ways, such as this notebook computer to develop the project. I would like to share the honor with them. I also want to thank all my friends who helped me during my studying at CSUSB.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGMENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER ONE: INTRODUCTION	
1.1 Purpose of the Project	1
1.2 Scope of the Project	4
1.3 Significance of the Project	5
1.4 Limitation of the Project	6
1.5 Definitions and Abbreviations	6
1.6 Organization of the Documentation	8
CHAPTER TWO: RELATED WORK REVIEW	
2.1 Introduction	9
2.2 Related Designs	10
CHAPTER THREE: SOFTWARE REQUIREMENTS SPECIFICATION	
3.1 Introduction	15
3.2 Overall Description	15
3.2.1 Product Perspective	15
3.2.2 Product Functions	20
3.2.3 User Characteristics	23
3.2.4 Constraints	23
3.2.5 Assumptions and Dependencies	23
3.3 Specific Requirements	24

3.3.1 External Interface Requirements	24
3.3.2 Functional Requirements	25
3.3.3 Performance Requirements	27
3.3.4 Software System Attributes	27
CHAPTER FOUR: PROBABILITY ANALYSIS	
4.1 Risk Probability in Special Patterns	28
4.1.1 One Number Known	28
4.1.2 Two Numbers Known	28
4.2 Simulation Results	44
4.3 Discussion	45
CHAPTER FIVE: SOFTWARE DESIGN	
5.1 Architecture Design	48
5.2 Detailed Procedures	59
5.3 Pseudo Code	69
CHAPTER SIX: MAINTENANCE MANUAL	
6.1 Source Files	76
6.2 Installation Description	77
CHAPTER SEVEN: CONCLUSIONS AND FUTURE DIRECTION	
7.1 Conclusions	78
7.2 Future Direction	79
APPENDIX A: SOURCE CODE OF AUTOMINE	81
APPENDIX B: SOURCE CODE OF SIMULATION	87
APPENDIX C: MAKEFILE	98
REFERENCES	100

LIST OF TABLES

Table 1. Definitions and Abbreviations	7
Table 2. Prolog Code of Minesweeper End Game	10
Table 3. Risk Simulated Results	45
Table 4. Calculated Probability	46
Table 5. State Variable of Xbomb	69
Table 6. Source Files	76

LIST OF FIGURES

Figure 1. The Minesweeper Game	3
Figure 2. Example of Local Probabilities	14
Figure 3. The Minesweeper Game	17
Figure 4. Deployment Diagram	18
Figure 5. Memory Constraints in the Project	20
Figure 6. Cases with Safe Cells	21
Figure 7. Cases with Risky Cells	21
Figure 8. Use Case Diagram	24
Figure 9. The Automine Shows Green for Safe	25
Figure 10. One Number Known	28
Figure 11. (1, 1) Two Cases	29
Figure 12. (1, 2) Two Cases	31
Figure 13. (1, 3) Two Cases	32
Figure 14. (1, 4) One Case	34
Figure 15. (2, 2) Three Cases	35
Figure 16. (2, 3) Three Cases	37
Figure 17. Results of Theory and Simulation	47
Figure 18. Initial Board of Beginner Level	48
Figure 19. Architecture of Xbomb	50
Figure 20. First Click	51
Figure 21. Enable Automine	52
Figure 22. Architecture of Automine	53
Figure 23. Intermediate Level	54

Figure 24. Expert Level	55
Figure 25. Risk of Special Pattern (1, 4)	56
Figure 26. Risk of Patterns (1, 1) and (1, 2)	57
Figure 27. Risk of Patterns (1, 2) and (3, 3)	58
Figure 28. Risk of Rotated Pattern (2, 3)	58
Figure 29. Initial Two Clicks	59
Figure 30. Choose the Smallest Risk	60
Figure 31. Show Ordered Items' Status	61
Figure 32. Marking Red Squares Would Not Help	62
Figure 33. Select the Smallest Risk Squares	63
Figure 34. Select Green	64
Figure 35. Need to Guess	65
Figure 36. Get More Greens	66
Figure 37. Only Red Squares Left	66
Figure 38. Game Won, Stop	67
Figure 39. The Highest Score of Beginner Level	68

CHAPTER ONE

INTRODUCTION

1.1 Purpose of the Project

The mine sweeper game can be viewed as a logic game or probability game. The object of Minesweeper is to find all the mines as quickly as possible without uncovering any of them. The player clicks any square on the playing field to uncover it. If it is a mine, the player loses the game. Otherwise, a number is displayed indicating how many mines are in the eight squares that surround the numbered one. Besides, clicking one of the safe areas will uncover the whole connected safe area. The player could right-click the square to mark it as a mine. Two right-clicks mark it with a question mark ('?'). Later, a player can either mark the square as a mine or uncover it by right-clicking again once or twice. When the player has marked all mines around a numbered square, the player can quickly uncover all empty squares around it by clicking that square with both mouse buttons. A good player looks for common patterns in numbers, which often indicate a corresponding pattern of mines.

The Minesweeper solver program (Automine) is based on the Linux xwindow C program with xwindow graphic library.

There will be a user interface that can let user to play the minesweeper game and keep the statistic of the good choice of next step for various patterns. There will be a heuristic demo mode that shows steps generated from different heuristics and the results of successful rate and time used will be added into the statistics. Besides, there will be just heuristic running mode to collect that information fast.

As shown in Figure 1, there are the numbers of mines not found and total time used. If the player clicked the save area, it will show all the safe area and the number of mines in the edge area. User can mark the mine with a red flag. However, if user marks the wrong one, it will show the "x" mark after the mine explores. The exploded mine will be red and the others will show up.

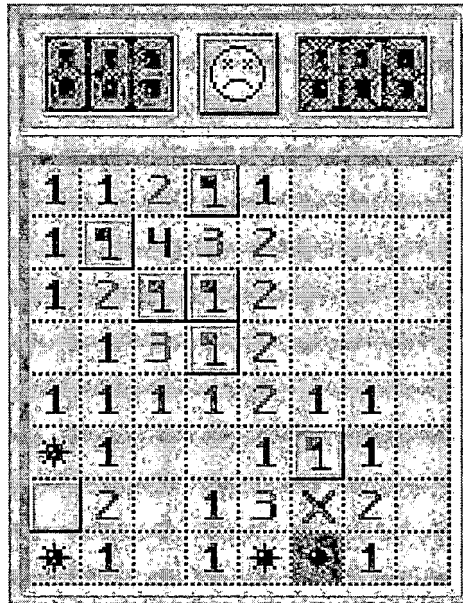


Figure 1. The Minesweeper Game

There are certain cases that you can decide if there is or is not a mine. However, there are also cases that you can not decide whether there is a mine or not and you have to guess. This project will develop heuristics to help users calculate the probabilities while achieving acceptable performance such as minimal time and successful rates.

This project applies three colors: red, yellow, and green to unknown squares for the risk probability. The red color represents the highest risk square which has definitely a bomb underneath, and the green color represents a safe square which has no bomb for sure, and the yellow color represents undecided squares which may or

may not have bombs. The size of the yellow maker is proportional to the risk. In the beginning, user has to randomly guess several steps to get the uncovered area as large as possible to achieve the minimal time performance. Then, the heuristic will find safe uncovered squares as well as squares with bombs and assign green and red to them respectively. Besides, this project calculates probability for each square which is neither red nor green. Furthermore, the probabilities of overlap conditions and specific patterns, which are derived or obtained from simulation, will be applied.

1.2 Scope of the Project

The software product defined in this document is to be known as heuristics for the minesweeper game. The delivery of this product will provide the following:

1. Description and implementation of the minesweeper game.
2. Write a plan on how and when to finish the project.
3. Information regarding the patterns that can decide if there is or is not a mine.
4. Special attention will be paid in guessing the next step if there is no determinable step to take.

5. Study and implement 3 heuristics to solve the minesweeper game.

5.1 Full Search - show the safe square and bomb.

5.2 Overlap Adjustment - Search for overlapped squares, increase its probability and reduce those of the others.

5.3 Special Pattern - Search for isolated special patterns and apply simulated probability.

6. The performance will be analyzed and compared in the successful rate, time required and resource required.

7. User can compete with the same game as the heuristics and the program can learn the most successful way to solve specific patterns.

8. All description and source codes will available in pure text, Post-Script or Adobe Acrobat PDF format.

1.3 Significance of the Project

This project will develop heuristics for the minesweeper game to solve the problem while achieving satisfaction performance. The program will help human to spot squares with bombs red and squares with no bomb green. Some patterns are applied to get more exactly probabilities to help user to determine the risk of a bomb. The brain works with the computer to achieve better performance.

1.4 Limitation of the Project

Although many squares can be decided to be safe or unsafe, there are still squares which have to be guessed. Here the program takes to tries to calculate the risk. However, a square with low risk may have a bomb underneath it. This makes the game more interesting!

1.5 Definitions and Abbreviations

The definition of special terminology and the acronyms as well as abbreviations are listed in Table 1.

Table 1. Definitions and Abbreviations

Automine	Provide colors to show risk in Xbomb game.
C, C++	High level programming language.
Heuristic	A simplified algorithm that finds near optimal solutions with reasonable time and resource instead of searching the whole solution space.
IEEE	Institute of Electrical and Electronics Engineers
Linux	A free operation system.
Minesweeper	A game distributed with Microsoft Windows. There are numbers on the rectangular grid board showing the number of mines in its 8 neighbors.
Prolog	Artificial Intelligence programming language.
Risky Square	Can't decide if there is a bomb.
SRS	Software Requirements Specification
Safe Square	No bomb for sure
Xbomb	Free minesweeper game in Linux.
Xwindow	Graphics programs read keystrokes and mouse clicks, and display shapes in Linux.

1.6 Organization of the Documentation

The remaining sections of this document will be organized as follows: Chapter 2 provides information regarding the related work. Chapter 3 describes the software requirement specification (SRS). Chapter 4 illustrates the probability analysis. Chapter 5 introduces the detailed software design. Chapter 6 is maintenance manual. Chapter 7 contains conclusions and points out future work.

CHAPTER TWO
RELATED WORK REVIEW

2.1 Introduction

Clay Mathematics Institute has provided Million Dollar for solving Minesweeper because it is an NP-complete problem [1].

In the paper of "Some Minesweeper Configuration", Dr. Richard Kaye proved that minesweeper is NP-complete and collected some interesting configurations. He mentioned that playing Minesweeper is about probabilities, not certainties. There are also variations of minesweeper games such as a 3-Dimensional version and infinite version [7].

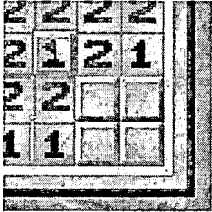
Dr. Hudelson's paper "The Math of Minesweeper" provided five theorems of the game and programmed these theorems into a solver. The results showed that the hexagonal shape was the easiest because each cell had only six adjacent cells. That would make the regions smaller and the theorems more powerful [3].

2.2 Related Designs

Professor Botting in California State University, San Bernardino, developed a minesweeper end game with Prolog [6]. If it is the last few steps, the one upper row and one left column should be known numbers otherwise the information could be used. If there is only one mine left, it must in W (1,0,0,0) to be shared with two '1's. If there are two mines left, they could be X and Y (0,1,1,0) or W and Z (1,0,0,1).

Table 2. Prolog Code of Minesweeper End Game

```
% A minesweeper end game
% In the last few steps of the minesweeper game, you
have a single mine to flag
% It is somewhere in the 4 squares in the bottom right
hand corner of the board
% The corner looks like this ( f is a flag, W, X, Y, Z
are unknown squares)
%   f   2   1
%   2   W   X
%   1   Y   Z
% where is the last mine?
```



```

% Encode W, X, Y, Z is 1 for a mine and 0 for no mine
on that square unique u(...) is true if precisely one
argument is 1 and the rest are 0
u(1,0,0,0), u(0,1,0,0), u(0,0,1,0), u(0,0,0,1).

% original problem
where(Field): -Field=[W, X, Y, Z], u(W,X;Y,Z)
    1 is W+X,
    2 is 1+W+X,
    2 is 1+W+Y,
    1 is W+Y.

% suppose we don't know how many mines are in the
corner
where2(Field): -Field=[W, X, Y, Z],
(X=1;X=0), (W=1;W=0), (Y=1;Y=0), (Z=1;Z=0)
    1 is W+X,
    2 is 1+W+X,
    2 is 1+W+Y,
    1 is W+Y.

```

```

% optimized with the unknown number of mines
where3(Field): -Field=[W, X, Y, Z],
(X=1;X=0), (W=1;W=0),
    1 is W+X,
    2 is 1+W+X,
(Y=1;Y=0),
    2 is 1+W+Y,
    1 is W+Y,
(Z=1;Z=0).

go: - where3(Field), count(Field), fail.
results:-counters(W,X,Y,Z), total(T),
    W1 is 100.0*W/T, write(W1), write('\t'),
    X1 is 100.0*X/T, write(X1), nl,
    Y1 is 100.0*Y/T, write(Y1), write('\t'),
    Z1 is 100.0*Z/T, write(Z1), nl.
reset:-retract(counter(_,_,_)),
assert(counters(0,0,0,0)), retract(total(_)),
assert(total(0)).

```

```
:-dynamic(counters/4).  
counters(0,0,0,0).  
:-dynamic(total/1).  
total(0).  
  
    Count(Field):-retract(total(T0)), T1 is T0+1,  
assert(total(T1)), retract(counters(W0,X0,Y0,Z0)),  
Field=[W,X,Y,Z], W1 is W0+W, X1 is X0+X, Y1 is Y0+Y, Z1  
is Z0+Z, assert(counters(W1,X1,Y1,Z1)).
```

For another design in the paper of Minesweeper: Advanced Tactics introduced some tactics in playing the game and described how to resolve local probability conflicts [4]. Author Sean Barrett wrote about the advanced tactics of minesweeper games. If user only checks the local probabilities, you can see that each of the squares in the marked mutually exclusive groups have a 50-50 chance of being a mine. The definition of local probabilities is that if the square has a '1' next to two unknown squares, each has a 50% chance of being a mine. His abstract method of computing probabilities is to run through all possible arrangements of mines, discard the ones that don't match the data we've collected, and count up the statistics for

each possible location. The time is exponential in the number of unknown squares.

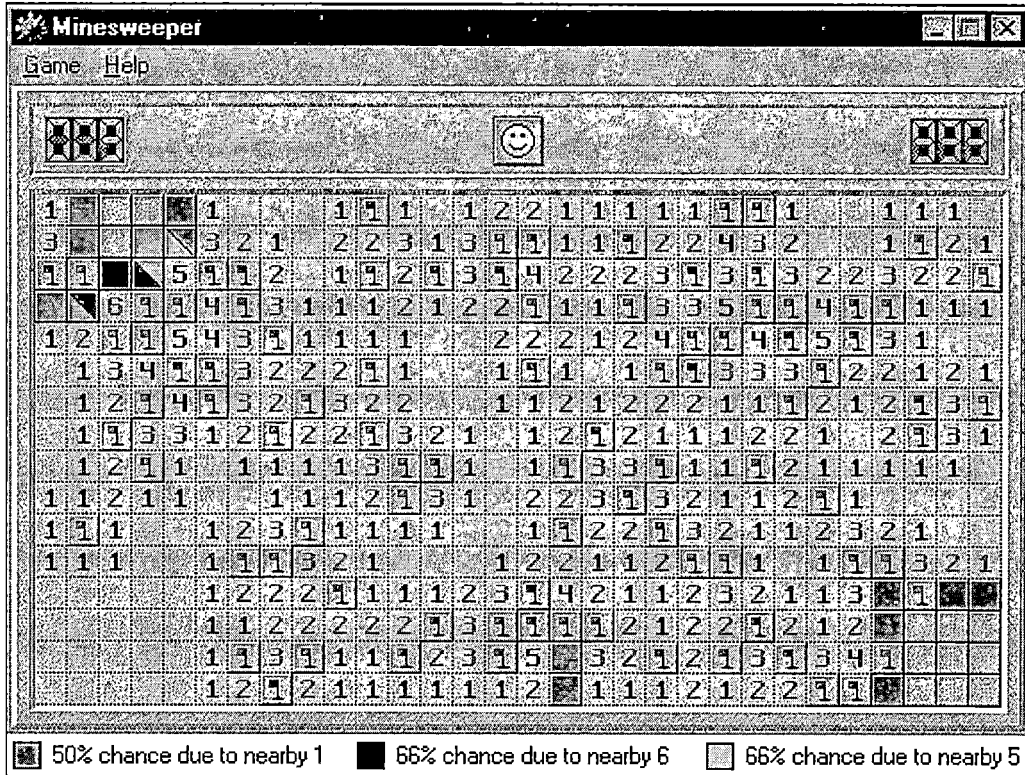


Figure 2. Example of Local Probabilities

Besides, there are many websites showing the description of advance tactics and strategies for solving the minesweeper [4,5].

CHAPTER THREE
SOFTWARE REQUIREMENTS SPECIFICATION

3.1 Introduction

This software requirement specification is for the automine project that helps to solve the popular minesweeper game by showing the hidden squares with green, red as well as yellow blocks whose size is proportional to the risk probability.

3.2 Overall Description

3.2.1 Product Perspective

The object of Automine is to add a helper to Minesweeper and find all the mines as quickly as possible without uncovering any of them. The player clicks any square on a rectangular playing field to uncover it. If it is a mine, the player loses the game. Otherwise, a number is displayed indicating how many mines are in the eight squares that surround the numbered one. Besides, clicking one of the safe area will uncover the whole connected safe area. The player could right-click the square to mark it as a mine. Two right-clicks mark it with a question mark ('?'). Later, a player can either mark the square as a mine or uncover it by right-clicking again once or twice. When

the player has marked all mines around a numbered square, the player can quickly uncover all empty squares around it by clicking that square with both mouse buttons. If not all mines touching the square are marked, the uncovered touching squares flash. A good player looks for common patterns in numbers, which often indicate a corresponding pattern of mines.

The automine solver program is base on the Unix Xwindow C program with xwindow graphic library. There will be a user interface that can let user to play the minesweeper game and keep the statistic of the good choice of next step for various patterns. There will be a heuristic demo mode that shows steps generated from different heuristics and the results of successful rate and time used will be added into the statistics. Besides, there will be a heuristic running mode to show information fast.

As shown in Figure 3, there are the number of mines not find and total time used. If the player clicked the safe area, it will show all the safe area and the number of mines in the edge area. User can mark the mine with a red flag. However, if user marks the wrong one, it will show the "x" mark after the mine explores. The exploded mine will be red and the others will show up.

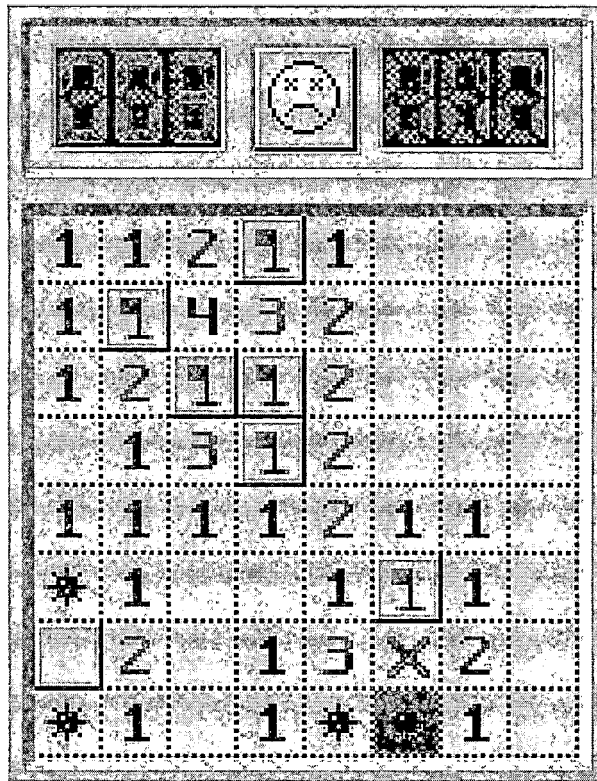


Figure 3. The Minesweeper Game

There are several modes for the Microsoft minesweeper game: 1. Beginner: 8x8 squares with 10 mines; 2. Intermediate: 16x16 squares with 40 mines; 3. Expert: 16x32 squares with 99 mines; and 4. Custom: Users define height, width, and number of mines. On a palm pilot PDA there are 11x11 squares with 20 mines.

3.2.1.1. The minesweeper solver will have a server to generate the new game and update the status from user/clients response as well as divide the problem and distribute to clients. The user can click a location to

uncover the number of mines in its neighbor or mark it as a mine. The clients will generate the next step according to the heuristics as shown in Figure 4.

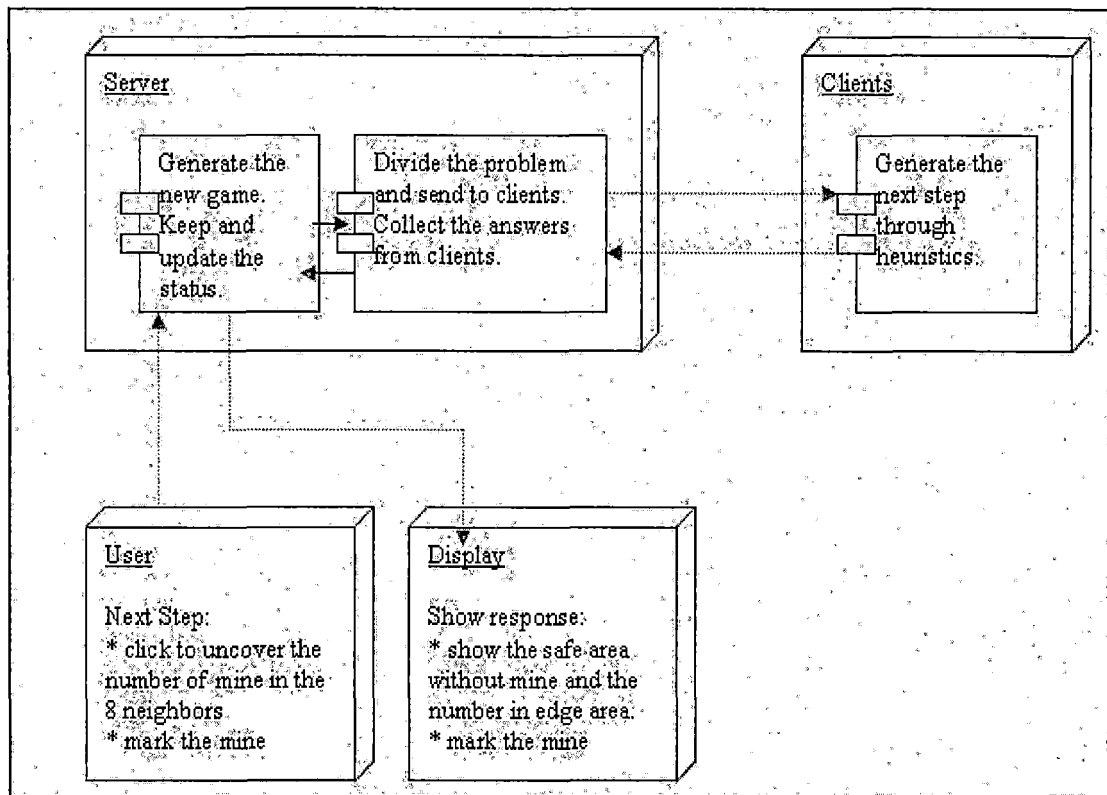


Figure 4. Deployment Diagram

3.2.1.2. The minesweeper solver (automine) will not implement hardware interface directly. However, it will trust the underlying operating system (Linux in this case) to handle the hardware interfaces.

3.2.1.3. The Graphic User Interface will be implemented through xwindow libraries. Users can use hot keys as well as mouse to play the game.

3.2.1.4. The software product detailed in this SRS would perhaps require more than 128 MB of Physical Memory depending on how large the game field is. The RAM needed by the product is essentially a function of the size of game, too. Memory constraints in the project are shown as figure 5.

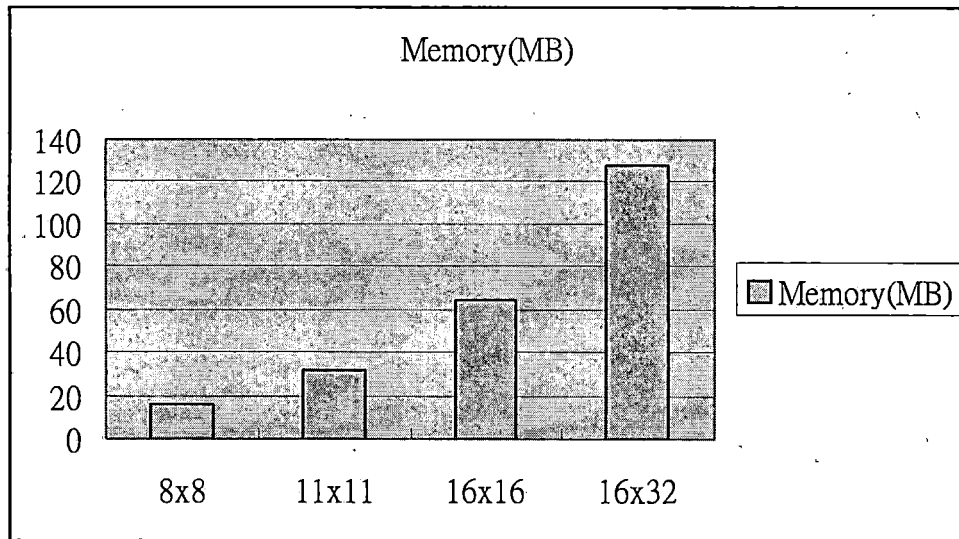


Figure 5. Memory Constraints in the Project

3.2.1.5. User chooses the New game from the Game Menu. There are four modes can be chosen: 1. Beginner; 2. Intermediate; 3. Expert. Follow the logic solution to uncover the safe squares and mark the mines until all mines are identified. The heuristic will generate the next steps and the successful rate and time used will be analyzed.

3.2.2 Product Functions

As detailed before (and repeated here for the reader's benefit), the product functions are as follows:

3.2.2.1. Description and implementation of the minesweeper game.

3.2.2.2. Write a plan on how and when to finish the project. It is designed for common users, expert users and software development programmers.

3.2.2.3. Information regarding the patterns that can decide if there is or is not a mine. For example,

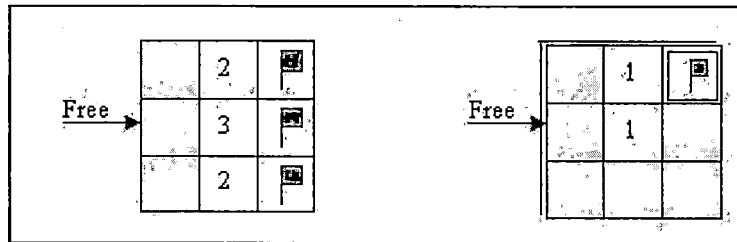


Figure 6. Cases with Safe Cells

3.2.2.4. Special attention will be paid in guessing the next step if there is no determinable step to take. For example, to guess the first square and second square. Or when there is no more information for deterministic step. One way is to try the safest square. The other way is to have some chances to take risks.

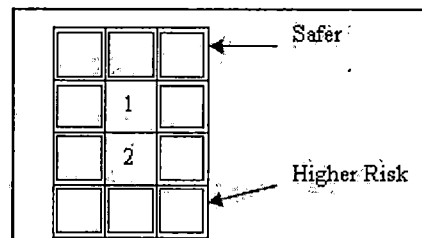


Figure 7. Cases with Risky Cells

3.2.2.5. Study and implement 3 heuristics to solve the minesweeper game. The initial idea is to implement (1) full search algorithm: try the first safe step; (2) overlap adjustment: find the overlap squares, increase the risk; (3) look up for special patterns: Find special patterns and apply simulation results of risk probability.

3.2.2.6. The performance will be analyzed and compared in the successful rate, time required and resource required. Sometimes, the competition only cares the minimal time to solve the problem. The heuristic may set the number of larger initial guessing steps to try its luck. If it unfortunately touches the mine, the heuristic can start over really quick. If it is lucky to get a large safe area, then it can divide the problem and solve it much faster. However, if the safety is the most important concern, the number of guessing steps will be minimized.

3.2.2.7. All description and source codes will available in pure text, Post-Script or Adobe Acrobat PDF format.

3.2.3 User Characteristics

The set of users can be grouped as follows:

- a. Common user: Use minesweeper solver to help playing the game. There are two types of users: "Risk Averse" users will spend time avoiding risky guesses and "speed demons" who will take risks to finish quickly. By the way the "Risk Averse" types tend to play a version that doesn't give a score—the Palm Pilot Mine Hunt for example. This software supports both!
- b. Software development programmer: Analyze the performance and learn from statistics.

3.2.4 Constraints

The original software keeps only the highest score. Here the top ten of each mode will be recorded.

3.2.5 Assumptions and Dependencies

Sometimes, there is an assumption that the first top-left square is safe for the game or at least first 2 squares in the top-left corner is uncovered to guarantee the game can be finished more safely.

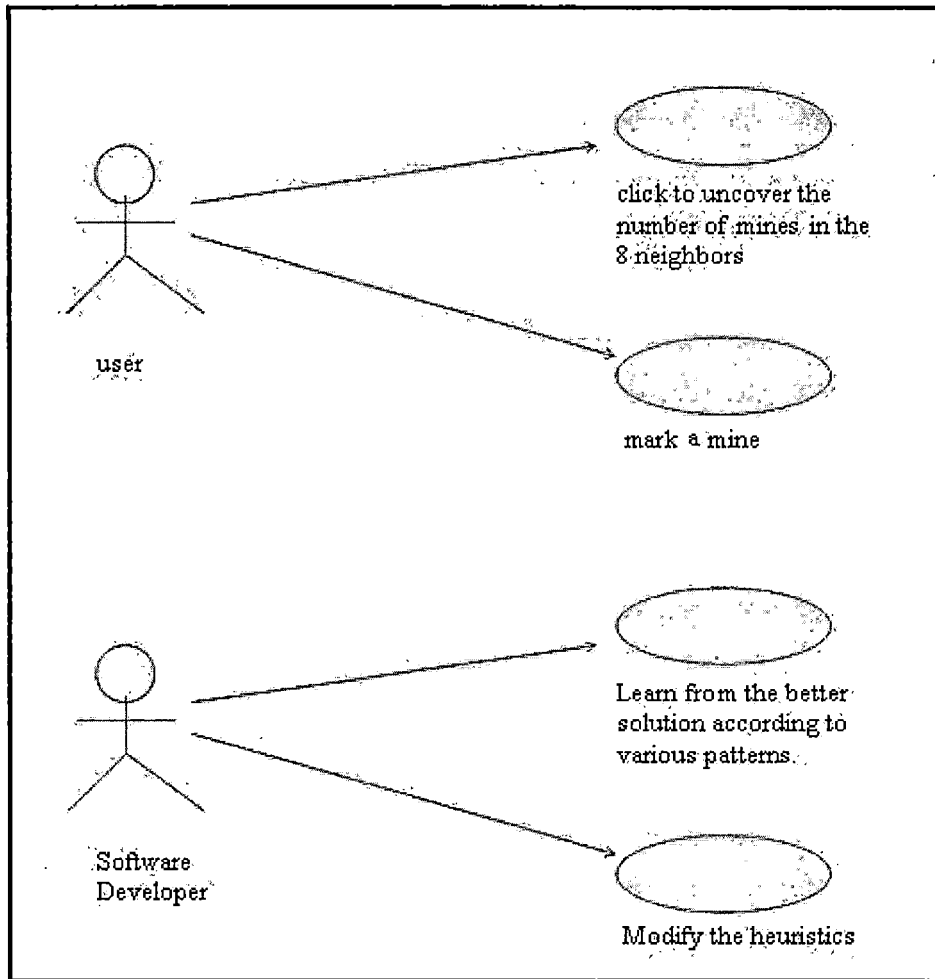


Figure 8. Use Case Diagram

3.3 Specific Requirements

3.3.1 External Interface Requirements

Some of these interfaces have been prototypes in C and xwindow library.

3.3.2 Functional Requirements

3.3.2.1. User can mark the mine and uncover the safe squares. If user wants to get help from the solver, click the Automine menu and the safe squares will become green, the square with bomb would become red and size of yellow markers shows the risk.

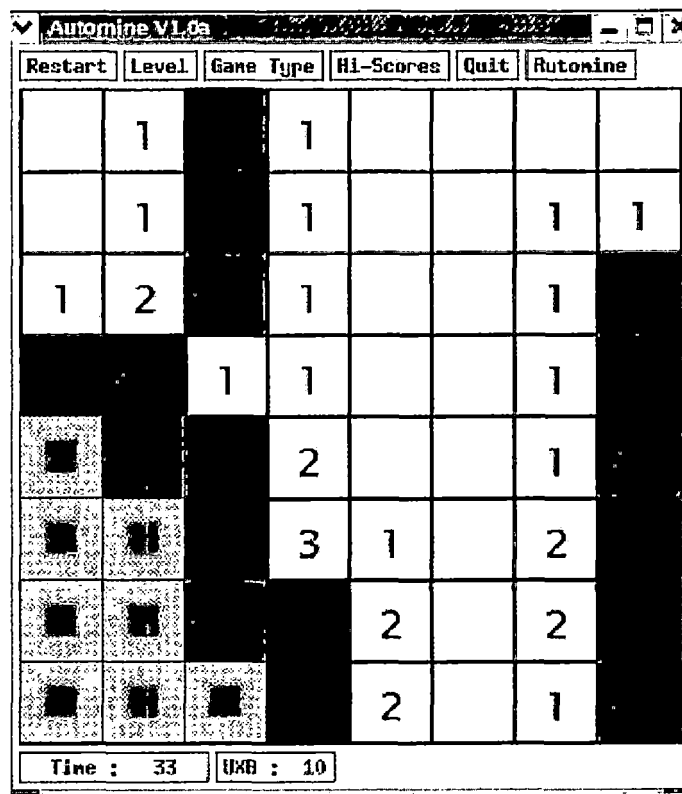


Figure 9. The Automine Shows Green for Safe

3.3.2.2. The display will include the number of mines left and the time used. The program will be initiated by the user and will remain active as long as the user desires to remain playing new games. The highest ten scores will be recorded.

Top 10 Mine Sweepers

Beginners:

1.	Richard	55 seconds	02/14/02	09:30:55
2.	Angela	60 seconds	03/18/02	15:20:05
.....				
10.	Tiffany	500 seconds	06/11/02	23:55:16

Intermediate:

1.	Richard	155 seconds	02/15/02	09:30:55
2.	Angela	160 seconds	03/19/02	15:20:05
.....				
10.	Tiffany	800 seconds	06/12/02	23:55:16

Expert:

1.	Richard	355 seconds	02/16/02	09:30:55
2.	Angela	460 seconds	03/16/02	15:20:05
.....				
10.	Tiffany	1800 seconds	06/13/02	23:55:16

3.3.3 Performance Requirements

The product should be running normally at all times. The highest score list can be reset. The statistic of various heuristic and criteria will be analyzed. For example:

3.3.4 Software System Attributes

The software development programmer should be able to use the system for the following reasons:

- a. Maintaining the system: remove the potential bugs in the system, improve the functions of the system based on user's feedback, answering questions, updating the necessary information.
- b. Security: encryption of the highest score record so that no one can edit it.
- c. Backup: frequently back up the system and the information. If there is any damage or change in the system or data, the backup can be restored to replace or compare the data or system.
- d. Check the good patterns used in the game and implement it in the database.

CHAPTER FOUR
PROBABILITY ANALYSIS

4.1 Risk Probability in Special Patterns

4.1.1 One Number Known

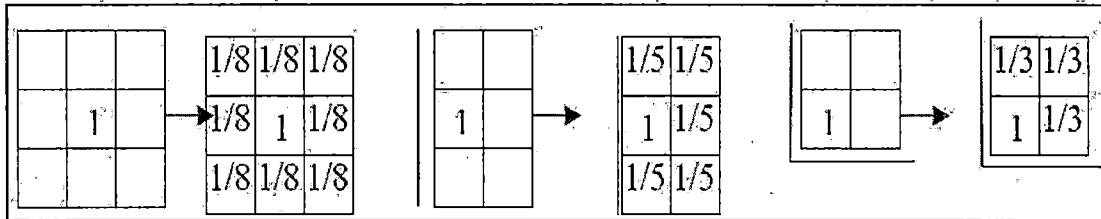


Figure 10. One Number Known

If there is only one number known, the probability of there is one mine in its neighborhood is $n/8$, where n is the known number. For the special case of the known number is at the corner or edge, the probability would become $1/3$ and $1/5$ respectively as shown in Figure 10.

4.1.2 Two Numbers Known

If there are two numbers known, the probability of there is one mine in its neighborhood is a very complicated conditional probability problem. The probability will be proportional to the combination of different situation and the size of board and the number of mines. For example, in the beginner level where $X=8$, $Y=8$, $N=10$, there are 64

squares in the board and there are $4 \times 3 = 12$ squares in the neighborhood of two known numbers and $64 - 12 = 52$ squares outside the neighborhood.

For known numbers (1,1), there are two cases: (A) two numbers share 1 mine and (B) two numbers share 0 mines. 'a' is the probability if one of the top 3 squares has bomb. 'b' is the probability if one of the 4 middle squares has bomb. 'c' is the probability if one of the 3 bottom squares has bomb. P(A) is the conditional probability when special pattern (1, 1) happens and case (A) happens. P(B) is the conditional probability when special pattern (1, 1) happens and case (B) happens.

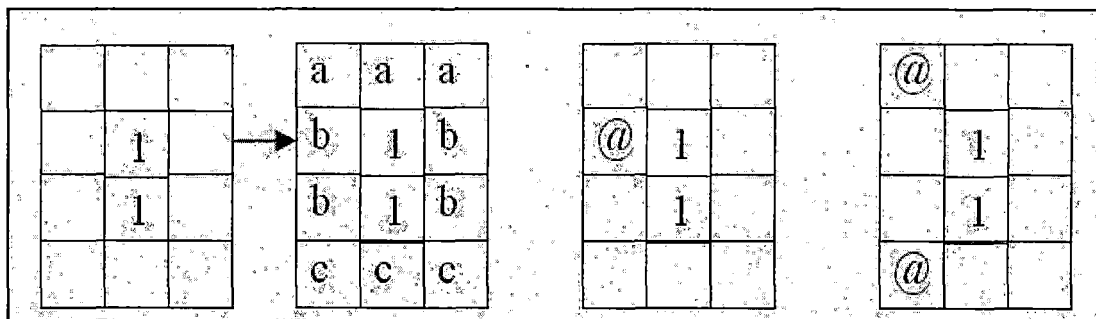


Figure 11. (1, 1) Two Cases

For case (A), there is one mine in the neighborhood and nine mines outside the neighborhood. The total combination is $C(4,1) \cdot C(52,9)$, where $C(m,n) = \frac{m!}{(m-n)! \cdot n!} = \frac{m \cdot (m-1) \cdot \dots \cdot (m-n+1)}{n!}$. For case (B), there are two

mines in the neighborhood and eight mines outside the neighborhood. The total combination is $C(3,1)*C(3,1)*C(52,8)$. The sum of probability of case A and B equals 1. Therefore, the probabilities can be calculated from the ratio of the two cases.

$$\begin{aligned}
 R = P(A)/P(B) &= \{C(4,1)*C(52,9)\} / \\
 &\{C(3,1)*C(3,1)*C(52,8)\} \\
 &= 4/9 * \{52*...*45*44/9!\} / \{52*...*45/8!\} \\
 &= 4/9 * 44/9 \\
 &= 2.1728
 \end{aligned}$$

$$P(A)+P(B)=1$$

$$R*P(B)+P(B)=1$$

$$P(B) = 1/(1+R) = 0.3152$$

$$P(A) = 1- P(B) = 0.6848$$

$$= P(B) * R$$

For case (A), four squares share the same probability of one mine. Therefore,

$$b = P(A)/C(4,1) = 0.1712.$$

For case (B), three squares share the same probability of one mine. Therefore,

$$a = c = P(B)/C(3,1) = 0.105.$$

For known numbers (1,2), there are two cases: (A) two numbers share 1 mine and (B) two numbers share 0 mine.

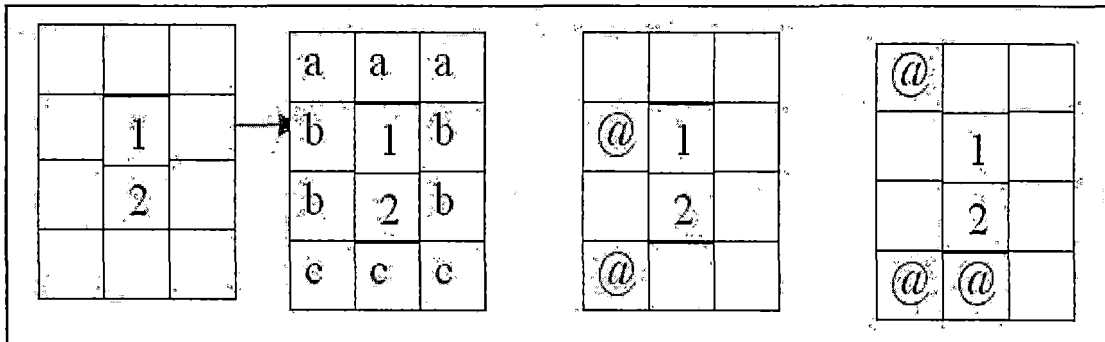


Figure 12. (1, 2) Two Cases

For case (A), there is two mine in the neighborhood and eight mines outside the neighborhood. The total combination is $C(4,1)*C(3,1)*C(52,8)$. For case (B), there are three mines in the neighborhood and seven mines outside the neighborhood. The total combination is $C(3,1)*C(3,2)*C(52,7)$.

The sum of probability of case A and B equals 1. Therefore, the probabilities can be calculated from the ratio of the two cases.

$$\begin{aligned}
 R = P(A)/P(B) &= \{C(4,1)*C(3,1)*C(52,8)\} / \\
 &\{C(3,1)*C(3,2)*C(52,7)\} \\
 &= 12/9*\{52*...*46*45/8!\}/\{52*...*46/7!\} \\
 &= 12/9*45/8 \\
 &= 7.5
 \end{aligned}$$

$$P(A)+P(B)=1$$

$$R*P(B)+P(B)=1$$

$$\begin{aligned}
P(B) &= 1/(1+R) = 0.1176 \\
P(A) &= 1- P(B) = 0.8824 \\
&= P(B) * R
\end{aligned}$$

For case (A), four squares share the same probability of one mine. Therefore,

$$b = P(A)/C(4,1) = 0.2204.$$

For case (B), three squares share the same probability of one mine.

Therefore,

$$a = P(B)/C(3,1) = 0.0392.$$

$$c = P(A)/C(3,1)+P(B)/C(3,2)*2= 0.3725$$

For known numbers (1,3), there are two cases: (A) two numbers share 1 mine and (B) two numbers share 0 mines (shown in Figure 13).

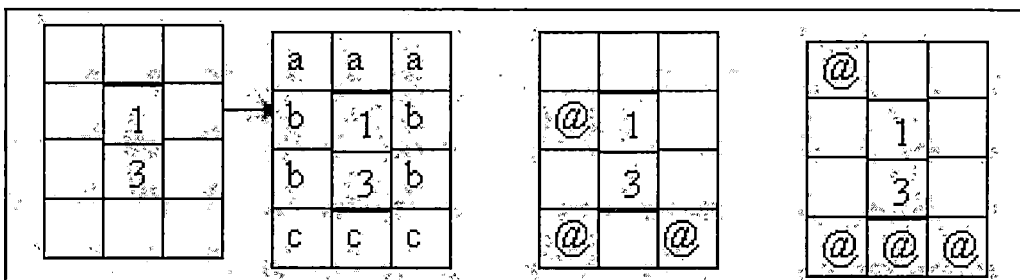


Figure 13. (1,3) Two Cases

For case (A), there is three mine in the neighborhood and seven mines outside the neighborhood. The total

combination is $C(4,1)*C(3,2)*C(52,7)$. For case (B), there are four mines in the neighborhood and six mines outside the neighborhood. The total combination is $C(3,1)*C(3,3)*C(52,6)$.

The sum of probability of case A and B equals 1. Therefore, the probabilities of A and B can be calculated from the ratio of the two cases.

$$\begin{aligned}
 R = P(A)/P(B) &= \{C(4,1)*C(3,2)*C(52,7)\} / \\
 &\{C(3,1)*C(3,3)*C(52,6)\} \\
 &= 12/3*\{52*...*47*46/7!\} / \{52*...*47/6!\} \\
 &= 12/9*46/7 \\
 &= 8.762
 \end{aligned}$$

$$P(A)+P(B)=1$$

$$R*P(B)+P(B)=1$$

$$P(B) = 1/(1+R) = 0.1024$$

$$P(A) = 1- P(B) = 0.8976$$

$$= P(B) * R$$

For b in case (A), four squares share the same probability of one mine. Therefore,

$$b = P(A)/C(4,1) = 0.2244.$$

For a in case (B), three squares share the same probability of one mine. Therefore,

$$a = P(B)/C(3,1)*1 = 0.0341.$$

For c in both case (A) and (B), three squares share the same probability of one mine. Therefore,

$$c = P(A)/C(3,2)*2+P(B)/C(3,3)=0.5984+0.1024=0.7008$$

For known numbers (1,4), there is only one case: (A) two numbers must share 1 mine.

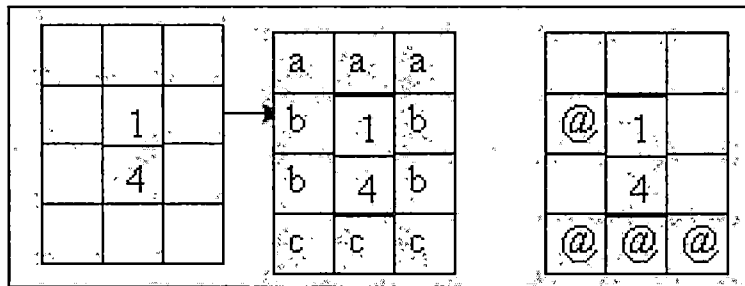


Figure 14. (1,4) One Case

For case (A), $P(A)=1$, $a=0$, $b = P(A)/C(4,1) = 0.25$, $c=1$.

For known numbers (2,2), there are three cases: (A) two numbers share 2 mines and (B) share 1 mine and (C) share 0 mines.

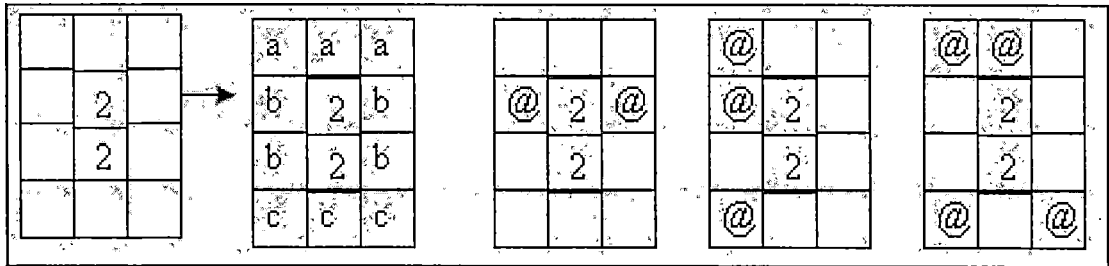


Figure 15. (2,2) Three Cases

For case (A), there are two mines in the neighborhood and eight mines outside the neighborhood. The total combination is $C(4,2)*C(52,8)$. For case (B), there are three mines in the neighborhood and seven mines outside the neighborhood. The total combination is $C(3,1)*C(4,1)*C(3,1)*C(52,7)$. For case (C), there are four mines in the neighborhood and six mines outside the neighborhood. The total combination is $C(3,2)*C(3,2)*C(52,6)$.

The sum of probability of case A, B and C equals 1. Therefore, the probability can be calculated from the ratio of the three cases.

$$\begin{aligned}
 R1 &= P(A)/P(C) = \{C(4,2)*C(52,8)\} / \\
 &\{C(3,2)*C(3,2)*C(52,6)\} \\
 &= 6/9*\{52*...*46*45/8!\} / \{52*...*47/6!\} \\
 &= 6/9*46*45/8/7 \\
 &= 24.64
 \end{aligned}$$

$$\begin{aligned}
R2 &= P(B)/P(C) = \{C(3,1) * C(4,1) * C(3,1) * C(52,7)\} / \\
&\{C(3,2) * C(3,2) * C(52,6)\} \\
&= 36/9 * \{52 * \dots * 47 * 46 / 7!\} / \{52 * \dots * 47 / 6!\} \\
&= 4 * 46 / 7 \\
&= 26.28
\end{aligned}$$

$$P(A) + P(B) + P(C) = 1$$

$$R1 * P(C) + R2 * P(C) + P(C) = 1$$

$$P(C) = 1 / (1 + R1 + R2) = 0.01926$$

$$P(A) = R1 * P(C) = 0.4746$$

$$P(B) = R2 * P(C) = 0.5062$$

For a, c in case (B) and (C),

$$a = c = P(B) / C(3,1) + P(C) / C(3,2) * 2 = 0.1816.$$

For b in case (A) and (B),

$$\begin{aligned}
b &= P(A) / C(4,2) * 3 + P(B) / C(4,1) = 0.2373 + \\
&0.1266 = 0.3639
\end{aligned}$$

(For example, (0011, 0101, 0110, 1001, 1010, 1100),
 $C(4,2) = 6$ cases, in which 3 cases have the first=1.)

For known numbers (2,3), there are three cases: (A) two numbers share 2 mines and (B) two numbers share 1 mine and (C) share 0 mines.

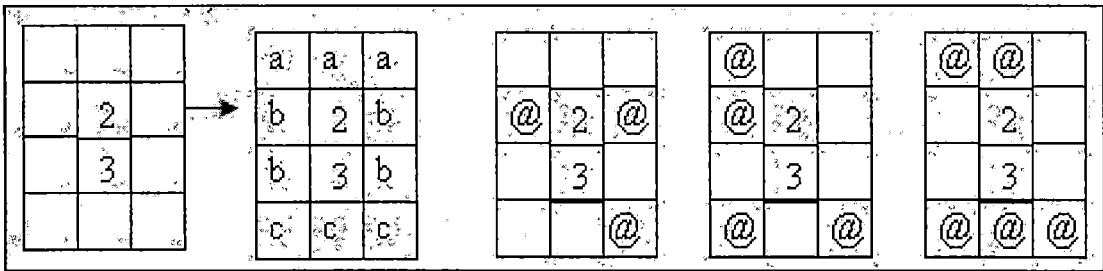


Figure 16. (2,3) Three Cases

For case (A), there are three mines in the neighborhood and seven mines outside the neighborhood. The total combination is $C(4,2)*C(3,1)*C(52,7)$. For case (B), there are four mines in the neighborhood and six mines outside the neighborhood. The total combination is $C(3,1)*C(4,1)*C(3,2)*C(52,6)$. For case (C), there are five mines in the neighborhood and five mines outside the neighborhood. The total combination is $C(3,2)*C(3,3)*C(52,5)$.

The sum of probability of case A and B equals 1. Therefore, it can be calculated from the ratio of the two cases.

$$\begin{aligned}
 R1 &= P(A)/P(C) = \{C(4,2)*C(3,1)*C(52,7)\} / \\
 &\{C(3,2)*C(3,3)*C(52,5)\} \\
 &= 18/3*\{52*...*47*46/7!\} / \{52*...*48/5!\} \\
 &= 18/3*47*46/7/6 \\
 &= 308.86
 \end{aligned}$$

$$\begin{aligned}
R2 &= P(B)/P(C) = \{C(3,1) * C(4,1) * C(3,2) * C(52,6)\} / \\
&\{C(3,2) * C(3,3) * C(52,5)\} \\
&= 36/3 * \{52 * \dots * 48 * 47 / 6!\} / \{52 * \dots * 48 / 5!\} \\
&= 36/3 * 47/6 \\
&= 94
\end{aligned}$$

$$P(A) + P(B) + P(C) = 1$$

$$R1 * P(C) + R2 * P(C) + P(C) = 1$$

$$P(C) = 1 / (1 + R1 + R2) = 0.002476$$

$$P(A) = R1 * P(C) = 0.7647$$

$$P(B) = R2 * P(C) = 0.2327$$

For a in case (B) and (C),

$$a = P(B) / C(3,1) + P(C) / C(3,2) * 2 = 0.0792.$$

For b in case (A) and (B),

$$\begin{aligned}
b &= P(A) / C(4,2) * 3 + P(B) / C(4,1) = \\
0.3824 + 0.0582 &= 0.4406.
\end{aligned}$$

(For example, (0011, 0101, 0110, 1001, 1010, 1100),
 $C(4,2) = 6$ cases, 3 cases have the first=1)

For c in both case (A), (B) and (C), therefore,

$$\begin{aligned}
c &= P(A) / C(3,1) + P(B) / C(3,2) * 2 + P(C) = \\
0.2549 + 0.1551 + 0.002476 &= 0.4125
\end{aligned}$$

For known numbers (2,4), there are three cases: (A) two numbers share 2 mines and (B) two numbers share 1 mine.

For case (A), there are 4 mines in the neighborhood and 6 mines outside the neighborhood. The total combination

is $C(4,2)*C(3,2)*C(52,6)$. For case (B), there are 5 mines in the neighborhood and 5 mines outside the neighborhood. The total combination is $C(3,1)*C(4,1)*C(3,3)*C(52,6)$.

The sum of probability of case A and B equals 1. Therefore, it can be calculated from the ratio of the two cases.

$$\begin{aligned}
 R = P(A)/P(B) &= \{C(4,2)*C(3,2)*C(52,6)\} / \\
 &\{C(3,1)*C(4,1)*C(3,3)*C(52,5)\} \\
 &= 18/12*\{52*...*47/6!\}/\{52*...*48/5!\} \\
 &= 18/12*47/6 \\
 &= 11.75
 \end{aligned}$$

$$P(A)+P(B)=1$$

$$R*P(B)+ P(B)=1$$

$$P(B) = 1/(1+ R) = 0.07843$$

$$P(A) = R*P(B) = 0.9216$$

For a in case (B),

$$a = P(B)/C(3,1) = 0.02614.$$

For b in case (A) and (B),

$$\begin{aligned}
 b &= P(A)/C(4,2)*3 +P(B)/C(4,1)= \\
 0.4608+0.01961 &= 0.4804.
 \end{aligned}$$

For c in both case (A), (B), therefore,

$$\begin{aligned}
 c &= P(A)/C(3,2)*2+P(B)/C(4,1)= \\
 0.6144+0.0784 &= 0.6928
 \end{aligned}$$

For known numbers (3,3), there are 4 cases: (A) two numbers share 3 mines and (B) share 2 mine and (C) share 1 mine (D) share 0 mines.

For case (A), there are three mines in the neighborhood and seven mines outside the neighborhood. The total combination is $C(4,3)*C(52,7)$. For case (B), there are four mines in the neighborhood and six mines outside the neighborhood. The total combination is $C(3,1)*C(4,2)*C(3,1)*C(52,6)$. For case (C), there are five mines in the neighborhood and five mines outside the neighborhood. The total combination is $C(3,2)*C(3,2)*C(4,1)*C(52,5)$. For case (D), $C(52,4)$.

The sum of probability of case A, B, C and D equals 1. Therefore, the probabilities can be calculated from the ratio of the four cases.

$$\begin{aligned}
 R1 &= P(A)/P(D) = \{C(4,3)*C(52,7)\} / C(52,4) \\
 &= 4*\{52*...*47*46/7!\} / \{52*...*49/4!\} \\
 &= 4*48*47*46/7/6/5 \\
 &= 1977
 \end{aligned}$$

$$\begin{aligned}
 R2 &= P(B)/P(D) = \{C(4,2)*C(3,1)*C(3,1)*C(52,6)\} / \\
 &C(52,4) \\
 &= 54*\{52*...*47/6!\} / \{52*...*49/4!\} \\
 &= 54*48*47/6/5 \\
 &= 4060.8
 \end{aligned}$$

$$\begin{aligned}
R3 &= P(C)/P(D) = \{C(3,2) * C(4,1) \\
&* C(3,2) * C(52,5)\} / C(52,4) \\
&= 36 * \{52 * \dots * 48 / 5!\} / \{52 * \dots * 49 / 4!\} \\
&= 36 * 48 / 5 \\
&= 345.6
\end{aligned}$$

$$P(A) + P(B) + P(C) + P(D) = 1$$

$$R1 * P(D) + R2 * P(D) + R3 * P(D) + P(D) = 1$$

$$P(D) = 1 / (1 + R1 + R2 + R3) = 0.00016$$

$$P(A) = R1 * P(D) = 0.3096$$

$$P(B) = R2 * P(D) = 0.6361$$

$$P(C) = R3 * P(D) = 0.0541$$

For a, c in case (B), (C) and (D),

$$\begin{aligned}
a=c &= P(B) / C(3,1) + P(C) / C(3,2) * 2 + P(D) = \\
&0.2483.
\end{aligned}$$

For b in case (A), (B) and (C),

$$\begin{aligned}
b &= P(A) / 3 + P(B) / C(4,2) * 3 + P(C) / C(4,1) = \\
&0.5637.
\end{aligned}$$

For known numbers (3,4), there are three cases: (A) two numbers share 3 mines and (B) share 2 mines and (C) share 1 mine.

For case (A), there are 4 mines in the neighborhood and 6 mines outside the neighborhood. The total combination is $C(4,3) * C(3,1) * C(52,6)$. For case (B), there are 5 mines in the neighborhood and 5 mines outside the neighborhood.

The total combination is $C(3,1)*C(4,2)*C(3,2)*C(52,5)$. For case (C), there are 6 mines in the neighborhood and 4 mines outside the neighborhood. The total combination is $C(4,1)*C(3,2)*C(52,4)$.

The sum of probability of case A, B and C equals 1. Therefore, the probabilities can be calculated from the ratio of the three cases.

$$\begin{aligned}
 R1 &= P(A)/P(C) = \{C(4,3)*C(3,1)*C(52,6)\} / \\
 &\{C(4,1)*C(3,2)*C(52,4)\} \\
 &= 12/12*\{52*...*48*47/6!\} / \{52*...*50*49/4!\} \\
 &= 12/12*48*47/6/5 \\
 &= 75.2
 \end{aligned}$$

$$\begin{aligned}
 R2 &= P(B)/P(C) = \{C(4,2)*C(3,1)*C(3,2)C(52,5)\} / \\
 &\{C(4,1)*C(3,2)*C(52,4)\} \\
 &= 18/12*\{52*...*48/5!\} / \{52*...*49/4!\} \\
 &= 54/12*48/5 \\
 &= 43.2
 \end{aligned}$$

$$P(A)+P(B)+P(C)=1$$

$$R1*P(C) + R2*P(C) + P(C) = 1$$

$$P(C) = 1/(1+R1+R2) = 0.00838$$

$$P(A) = R1*P(C) = 0.6298$$

$$P(B) = R2*P(C) = 0.3618$$

For a in case (B) and (C),

$$a = P(B)/C(3,1) + P(C)/C(3,2)*2 = 0.1262.$$

For b in case (A) and (B) and (C),

$$\begin{aligned} b &= P(A)/C(4,3)*3 + P(B)/C(4,2)*3 + P(C)/4 \\ &= 0.6554. \end{aligned}$$

(For example, (0011,0101,0110,1001,1010,1100),

C(4,2)=6 cases, 3 cases have the first=1)

For c in both case (A), (B) and (C), therefore,

$$\begin{aligned} c &= P(A)/C(3,1)+P(B)/C(3,2)*2+P(C) \\ &= 0.4600 \end{aligned}$$

4.2 Simulation Results

To verify the probability of these special patterns, a simulation utility program is developed. First, generate the board and randomly assign the mines. Second, count the total matched pattern and the total mines in each neighborhood location.

Pseudo code of `sim_mine.c`

```
// Minesweeper Utility
Define variable for beginner, medium, expert
main()
{
// random seed
    srand48((int)time());
// initialization
    new_board();
// define the special pattern
    for(i=0;i<Row;i++)
        for(j=0;j<Col;j++)
        {
//check in 4 directions to see if there is a hit
            hit();
// calculate the total mines in each location
            sum();
        }
}
```

// report: show probability of each location

The simulation result is shown in Table 1. The assumption of this simulation is other squares are hidden except the known pattern. This would be a good approximation if not much area is uncovered. So the result is still good for guessing the safe square.

Table 3. Risk Simulated Results

0.1069	0.1050	0.1039	0.0000	0.0000	0.0000	0.0270	0.0242	0.0299
0.1711	1	0.1716	0.2526	1	0.2500	0.4879	2	0.4901
0.1707	1	0.1707	0.2422	4	0.2552	0.4580	4	0.4828
0.1043	0.1052	0.1064	1.0000	1.0000	1.0000	0.7104	0.6896	0.6811
0.0382	0.0387	0.0388	0.1820	0.1806	0.1789	0.2427	0.2504	0.2488
0.2217	1	0.2220	0.3664	2	0.3627	0.5685	3	0.5664
0.2203	2	0.2203	0.3622	2	0.3672	0.5612	3	0.5619
0.3709	0.3768	0.3680	0.1775	0.1806	0.1835	0.2497	0.2462	0.2460
0.0119	0.0110	0.0144	0.0828	0.0779	0.0792	0.1430	0.1237	0.1201
0.2350	1	0.2396	0.4388	2	0.4357	0.6795	3	0.6657
0.2424	3	0.2458	0.4435	3	0.4422	0.6385	4	0.6295
0.6685	0.6937	0.6750	0.4143	0.4194	0.4062	0.4762	0.4418	0.4689

4.3 Discussion

Compare to the calculation, the simulation result is very good. The maximal variance of simulation result is about 0.04. However, the difference between the calculated probability and the average of simulation result is within +/- 0.0011. Therefore, the simulation and calculation is well matched.

Table 4. Calculated Probability

0.1050	0.1050	0.1050	0.0000	0.0000	0.0000	0.0270	0.0270	0.0270
0.1712	1	0.1712	0.2500	1	0.2500	0.4797	2	0.4797
0.1712	1	0.1712	0.2500	4	0.2500	0.4797	4	0.4797
0.1050	0.1050	0.1050	1.0000	1.0000	1.0000	0.6937	0.6937	0.6937
0.0392	0.0392	0.0392	0.1816	0.1816	0.1816	0.2473	0.2473	0.2473
0.2204	1	0.2204	0.3639	2	0.3639	0.5645	3	0.5645
0.2204	2	0.2204	0.3639	2	0.3639	0.5645	3	0.5645
0.3725	0.3725	0.3725	0.1816	0.1816	0.1816	0.2473	0.2473	0.2473
0.0122	0.0122	0.0122	0.0792	0.0792	0.0792	0.1289	0.1289	0.1289
0.2409	1	0.2409	0.4406	2	0.4406	0.6533	3	0.6533
0.2409	3	0.2409	0.4406	3	0.4406	0.6533	4	0.6533
0.6789	0.6789	0.6789	0.4125	0.4125	0.4125	0.4623	0.4623	0.4623

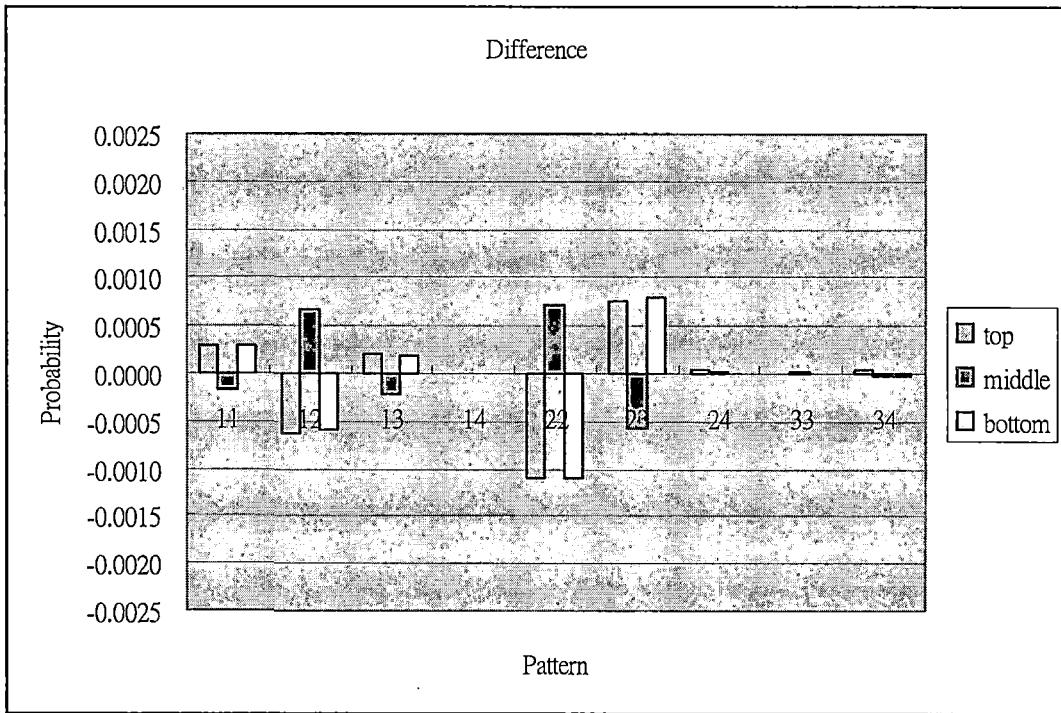


Figure 17. Results of Theory and Simulation

CHAPTER FIVE
SOFTWARE DESIGN

5.1 Architecture Design

In the automine project, the FindRisk() function is added to the basic xbomb program. The DrawRisk() function is added into the xwindow.c. Automine menu is added to toggle between enable/disable the helper. Figure 18 shows the architecture of the xbomb program and xwindow program as well as the modification of automine algorithms. In the left bottom corner, there are two blocks showing the time used and the bombs left.

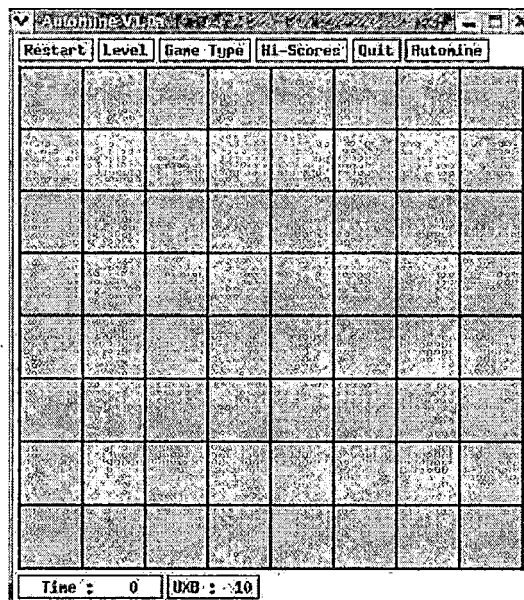


Figure 18. Initial Board of Beginner Level

The architecture of the original xbomb is shown in Fig 19. When the square is chosen, if it is a bomb, the game is over; if it is in the safe area (0 bomb in its neighborhood), it will uncover all the connected safe area.

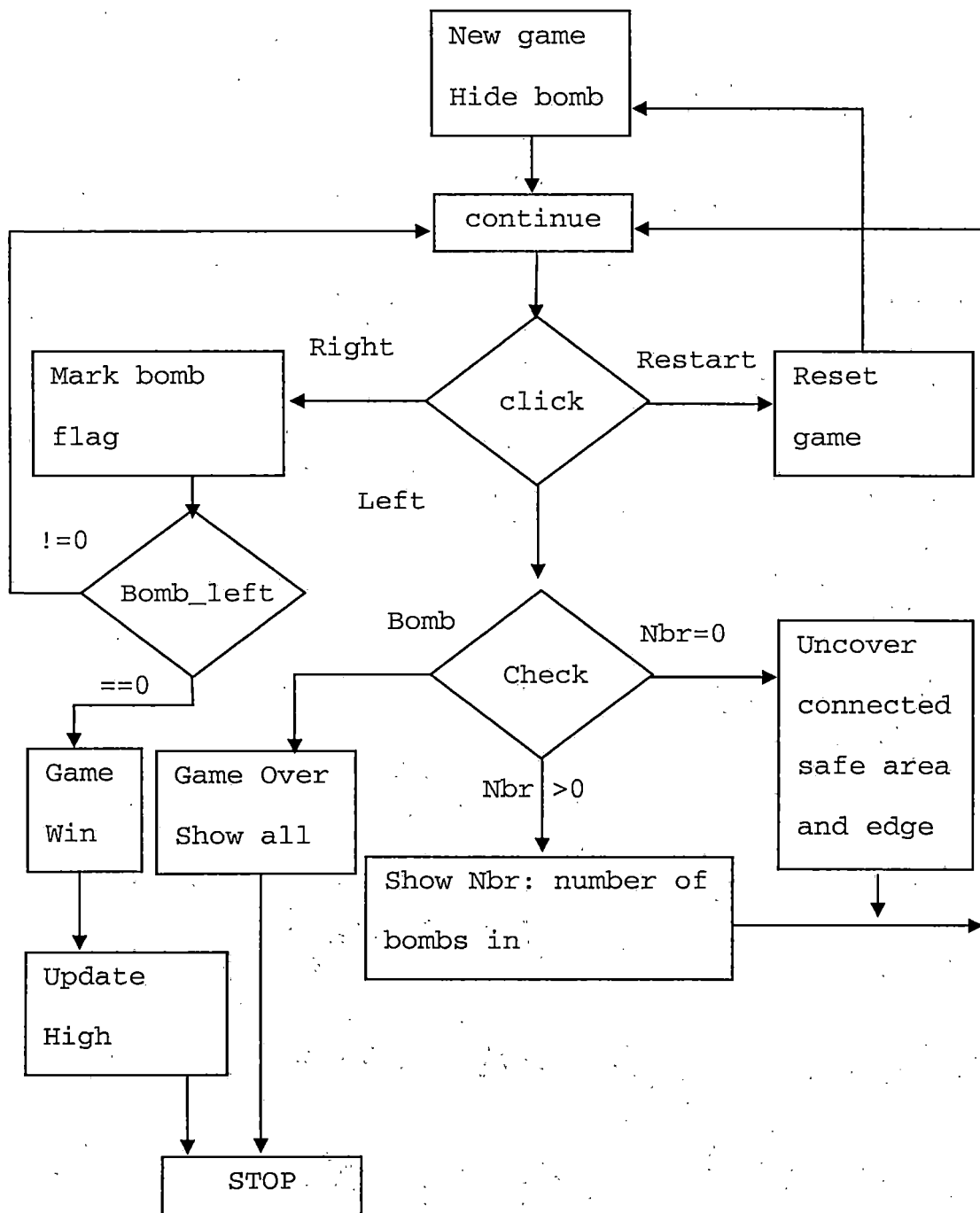


Figure 19. Architecture of Xbomb

In the automine project, the goal is to help users to achieve minimal time performance easily. The user should first randomly click several times to make the uncovered area as large as possible, more than 50%. It could hit bomb but speed is critical to achieve the minimal time.

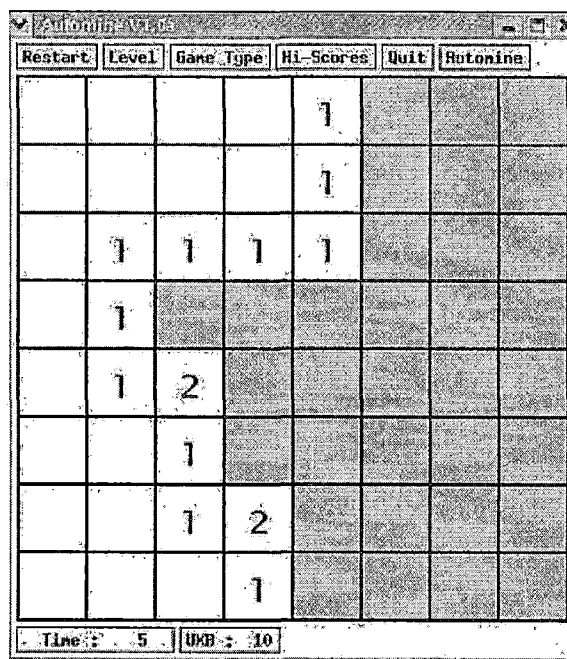


Figure 20. First Click

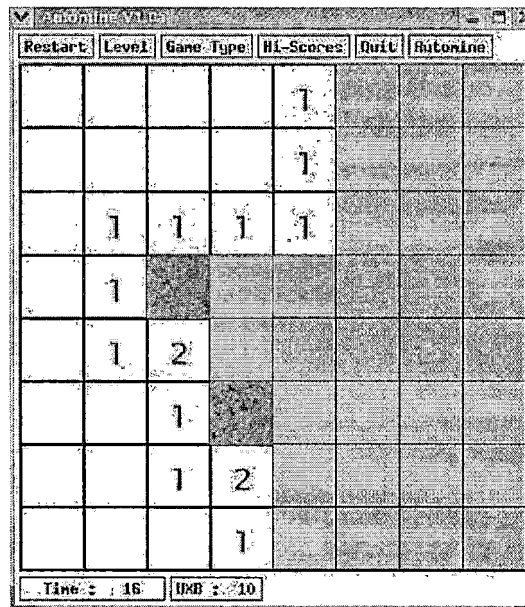


Figure 21. Enable Automine

After there is a large uncovered area, automine would show colors in hidden squares: green means absolutely safe, red means a bomb, yellow means unsure condition with the area proportional to the probability of risk. In many case, just follow the sequence to uncover or mark the closest green or red squares, the more information would lead to more certain decisions. However, there are still some cases that users need to guess when no green or red squares available. Users are suggested to choose the smallest risk. Keep in mind that sometimes the smallest risk one would still have a bomb hidden there.

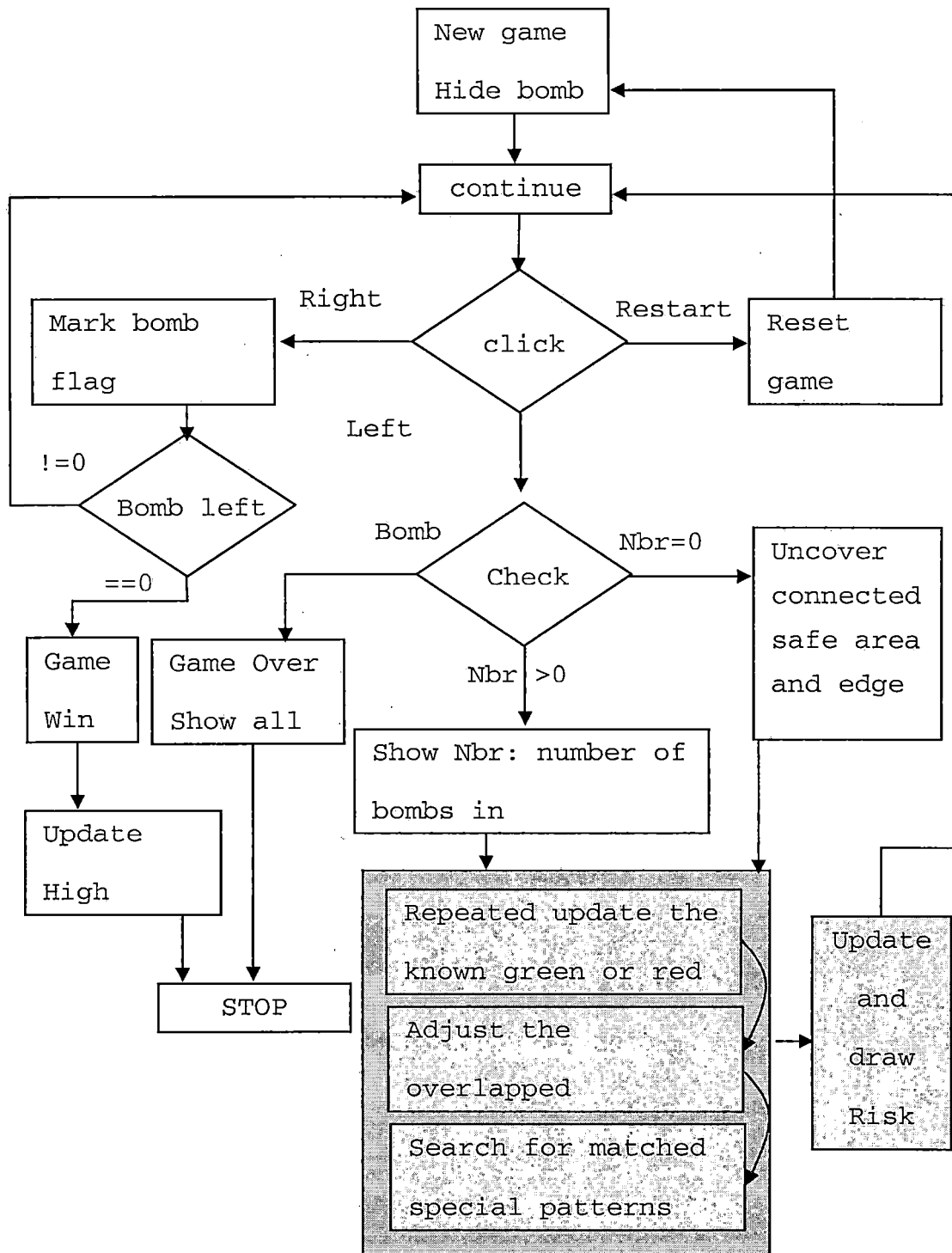


Figure 22. Architecture of Automine

The project covers the three typical levels: beginner (8x8, 10 bombs), intermediate (16x16, 40 bombs) and difficult (16x30, 100 bombs).

Autonine VLE																			
Restart				Level				Game Type				Hi-Scores				Quit			
																2	1		
1	1	3														3	2		
		1														2	2		
1	2	2														1	2		
																2	1		
1	2	3		2	2	1	2	2								1	2		
		2		2				1				3	2	1		2			
		2		2				1	1	1		1	1	2	1				
		2		2										1		1			
		2		2	1	1	1							1	1	1			
		2					2		1	1	1								
		1	1	1	2		2		1			1	1	1	1				
					1		1	1									1		
					1		2	1	2								1		
1	1				1											1	1		
		1			1														

Time: 9 UXB: 40

Figure 23. Intermediate Level

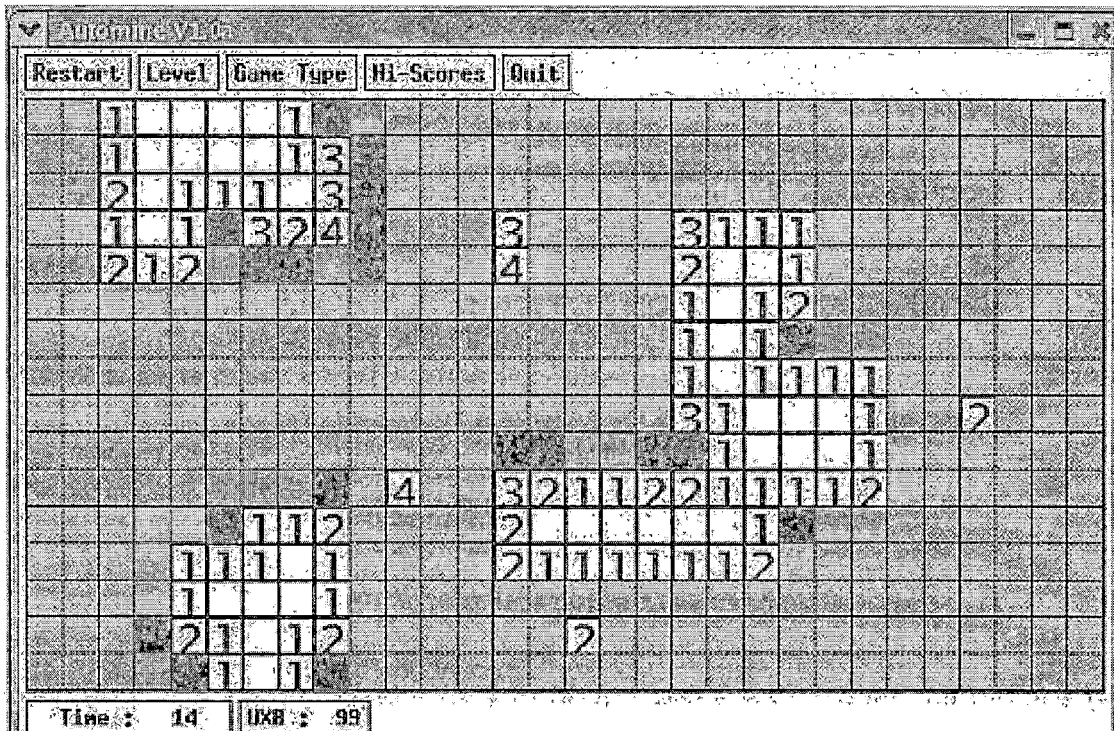


Figure 24. Expert Level

Three algorithms are applied:

1. Full search: to find out the green and red ones according to the known information.
2. Adjustment of overlap probability: When the hidden square has two or more neighbors uncovered, the risk probability calculated with the individual information would be adjusted. For example, the overlap one would take the maximum risk and the others should be reduced.
3. Special Patterns: There are certain patterns whose probability could be calculated or simulated. With

a table look up scheme, the probability of the neighbor squares could be identified precisely.

There are some special cases that the automine does not use repeated checking. For example, the special pattern matching happens in the latest stage of the three algorithms. The (4,1) pattern has the red and green squares but it seldom happens. Note: the overlap adjustment has the limitation that does not change the red and green status but just change the probabilities.

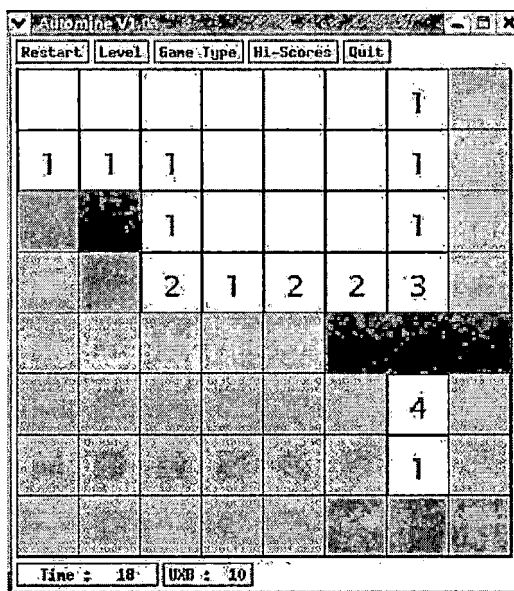


Figure 25. Risk of Special Pattern (1,4)

Figure 25 shows the repeated updating of the FindRisk status. The first one that can be determined is square (2,2) that has one bomb in its neighborhood and has only one

covered square. So the square must have a bomb and shown red color. After knowing that one is a bomb, the square (3,2) and (2,3) could determine the other squares are safe and shown green. So the process will be repeated until there is no more change.

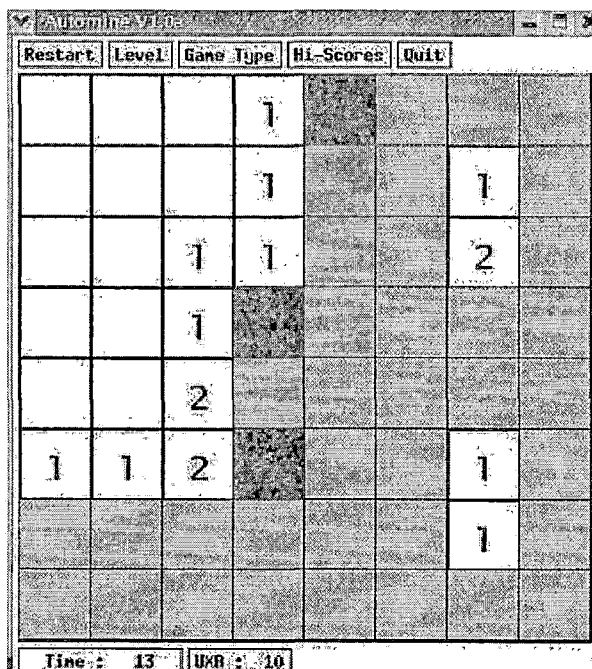


Figure 26. Risk of Patterns (1,1) and (1,2)

Figures 26, 27, 28 show the probability for special patterns.

A screenshot of a game window titled "Abandonia". The window contains a 7x7 grid. The top row has a button "Restart", a "Level" dropdown set to "1", and buttons "Game Type", "Hi-Scores", and "Quit". The grid contains the following numbers:
Row 1: 1, empty, empty, empty, empty, empty, empty
Row 2: 1, empty, empty, empty, empty, empty, empty
Row 3: 1, 1, empty, empty, empty, empty, empty
Row 4: empty, empty, empty, empty, empty, empty, empty
Row 5: empty, empty, 3, empty, empty, 2, empty
Row 6: empty, empty, 3, empty, empty, 1, empty
Row 7: empty, empty, empty, empty, empty, empty, empty
The bottom of the window shows "Time: :18" and "OMB: :10".

Figure 27. Risk of Patterns (1,2) and (3,3)

A screenshot of a game window titled "Abandonia". The window contains a 7x7 grid. The top row has a button "Restart", a "Level" dropdown set to "1", and buttons "Game Type", "Hi-Scores", and "Quit". The grid contains the following numbers:
Row 1: empty, 1, empty, empty, empty, empty, empty
Row 2: 1, 2, empty, empty, empty, empty, empty
Row 3: empty, empty, empty, 3, 2, empty, empty
Row 4: empty, empty, empty, empty, empty, empty, empty
Row 5: empty, empty, 2, empty, empty, 1, empty
Row 6: empty, empty, 2, empty, empty, 1, empty
Row 7: empty, empty, empty, empty, empty, empty, empty
The bottom of the window shows "Time: :14" and "OMB: :10".

Figure 28. Risk of Rotated Pattern (2,3)

5.2 Detailed Procedures

Here is an example of the expert level to illustrate the detailed procedures to use the automine game. The expert level here has 16x30 square board and 100 mines. The density of mine is higher than that of the beginner and medium level which has 8x8 board with 10 mines and 16x16 board with 40 mines. Figure 29 shows the result of the first 2 clicks. The safe area is not large enough. To achieve the best time performance, user needs to take the risk with more random clicks. Clicking green squares also gives more information for the risk averse player.

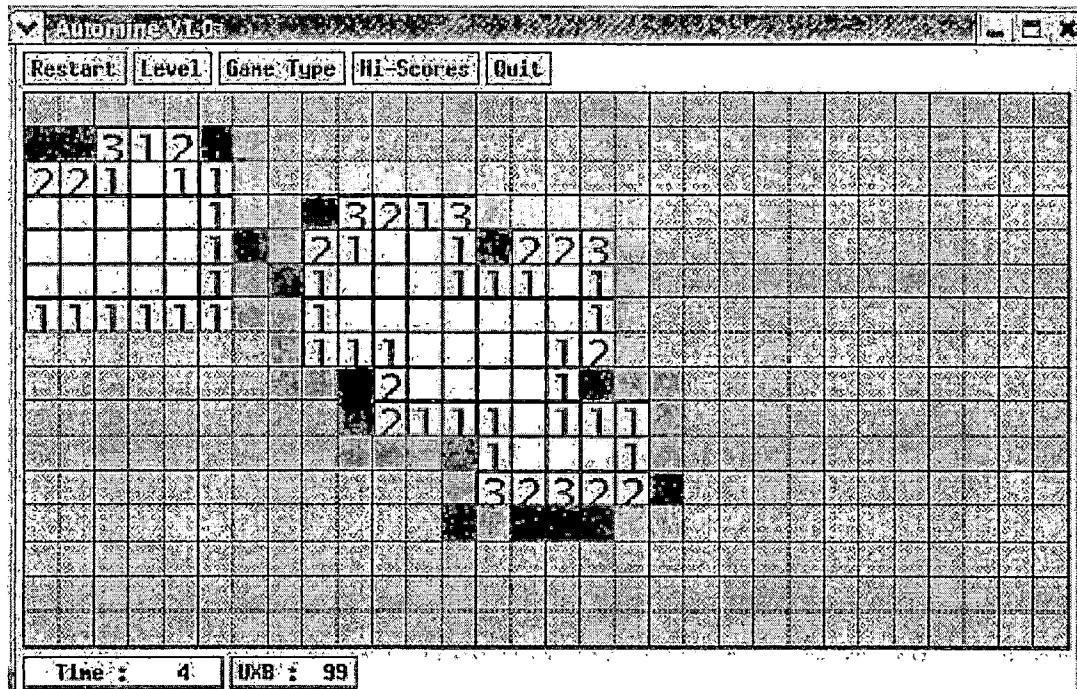


Figure 29. Initial Two Clicks

User can click on separated locations. Because the automine provides the precise probability with special pattern matching, it is suggested to click the adjacent squares. Figure 30 shows more clicks and there are special patterns (1,2) and (2,3). The risk probability below the (2,1) pattern is very small, so user can choose those quickly.

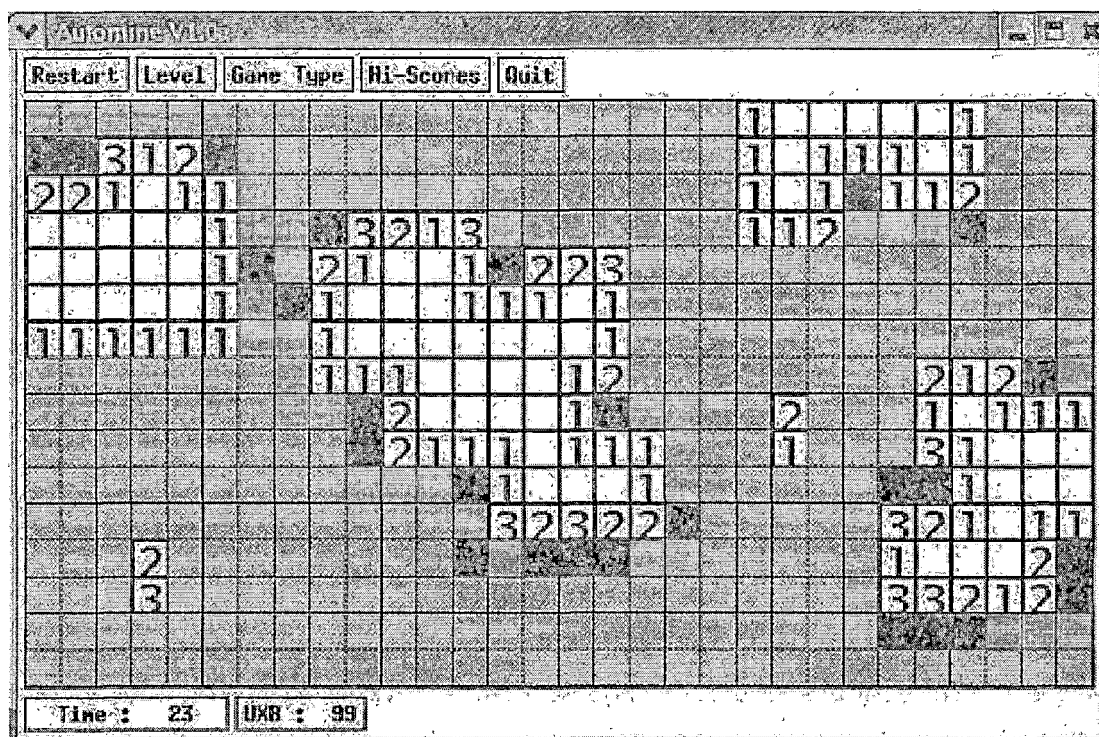


Figure 30. Choose the Smallest Risk

Figure 31 shows the results after choosing the small risk square below the special pattern (1,2). So those are really safe. User can see the risk area of the adjacent squares changed. However, those risks look similar because there is no enough information to tell the difference. If the probability is not making a big difference, it is high risk just to choose a small risk one. So it is time to uncover the know square to get more information.

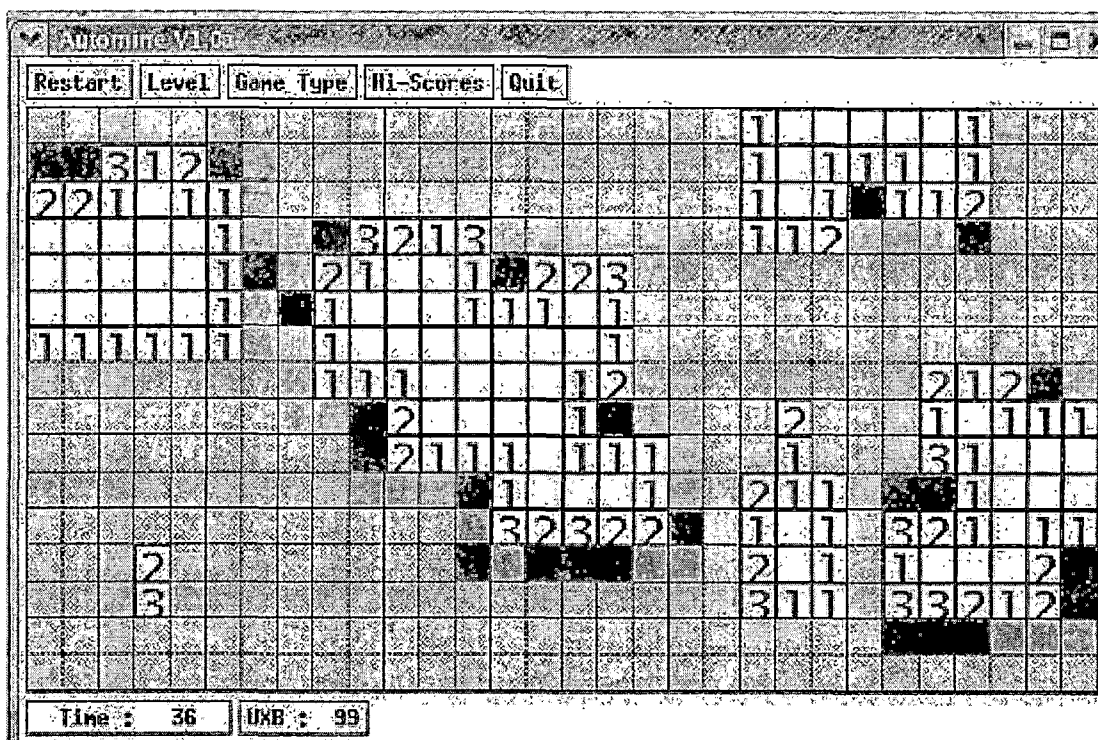


Figure 31. Show Ordered Items' Status

Here is a quick illustration that mark on the red square which indicate there is a bomb is not helping to get more information. Because, the automine use the iteration algorithm to check all useful information. Only when green square is uncovered and the information in the square would tell how many mines in its neighborhood, then the automine get more information.

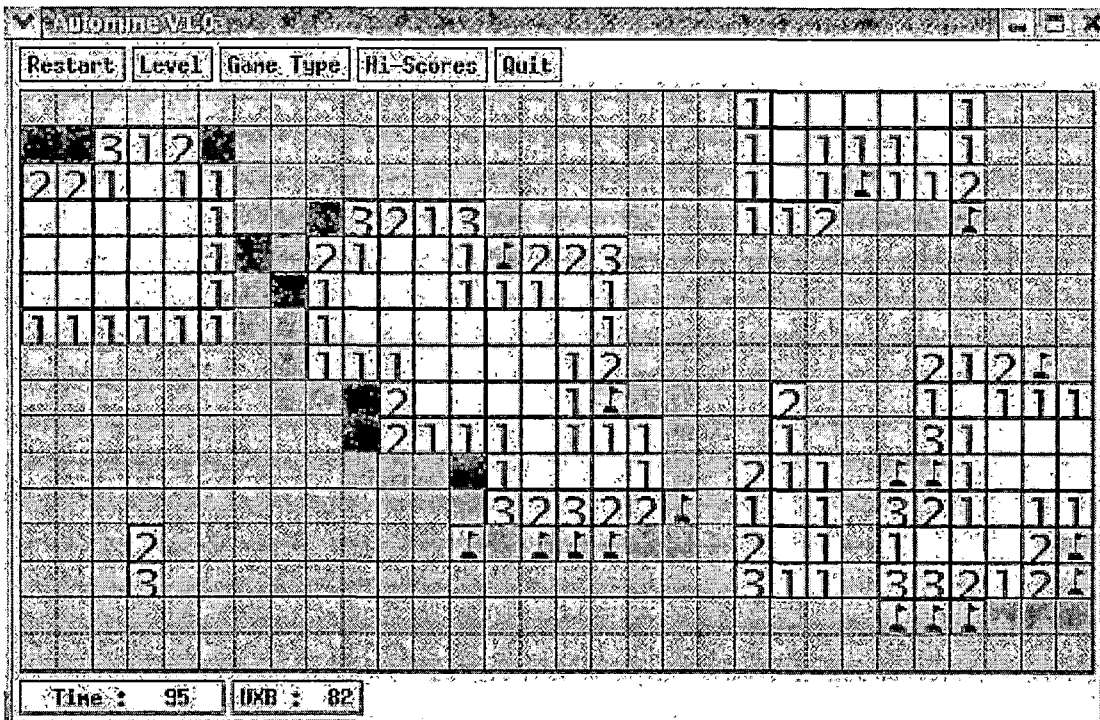


Figure 32. Marking Red Squares Would Not Help

There is a special pattern (2,3) that shows the small risk squares. User can select those and would get good results as shown in Figure 33.

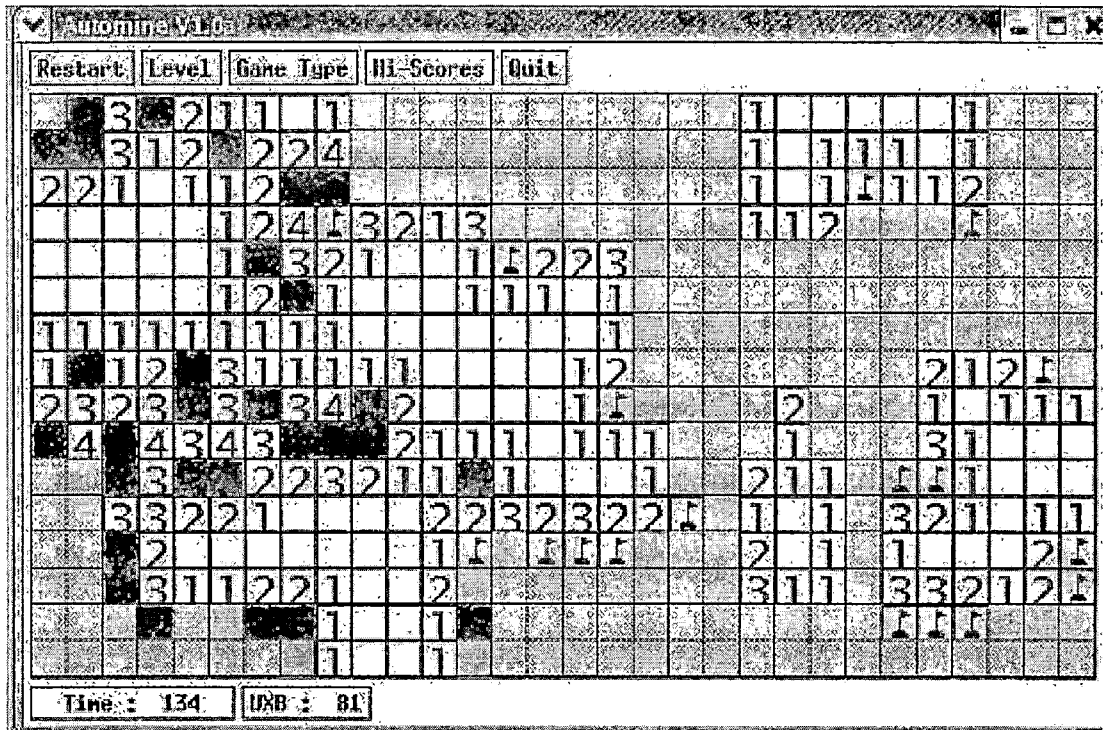


Figure 33. Select the Small Risk Squares

So the safe area is quite large now and user can begin to select all the green squares so that that information can be used to get more green and red squares as shown in Figure 34.

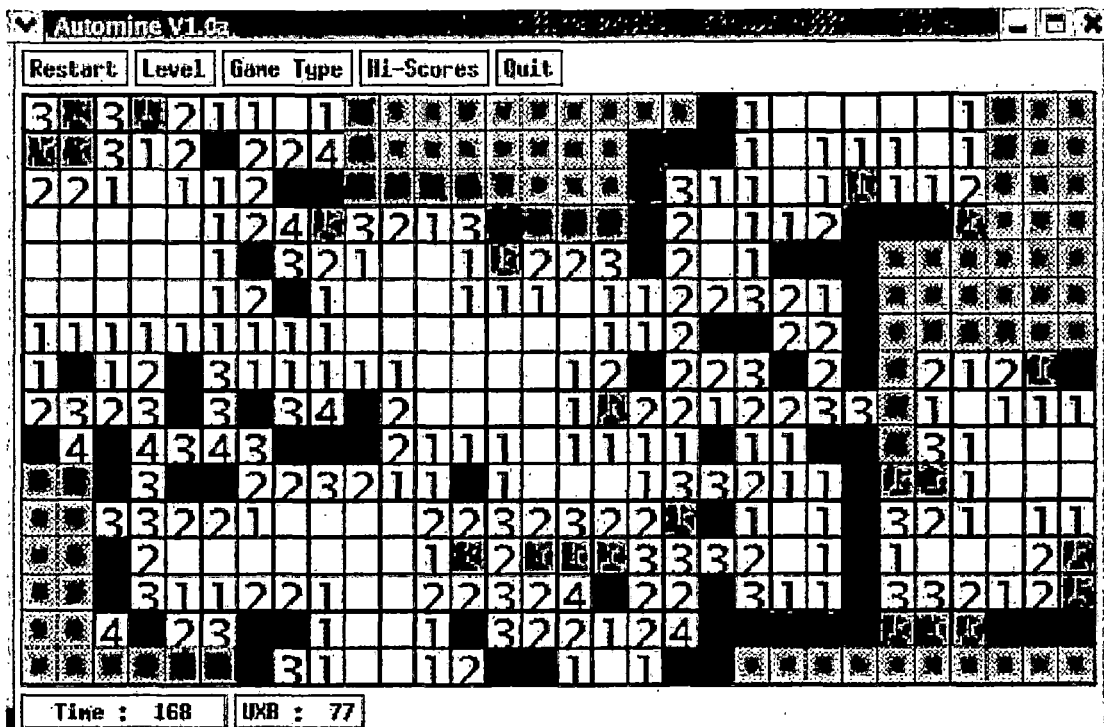


Figure 34. Select Green

It is very possible that all the green squares are selected but user needs to guess by the risk probability as shown in Figure 35. User can either guess the high risk one is a bomb or guess the small risk one is safe. Figure 36 shows more green squares appear as the result of a lucky guess.

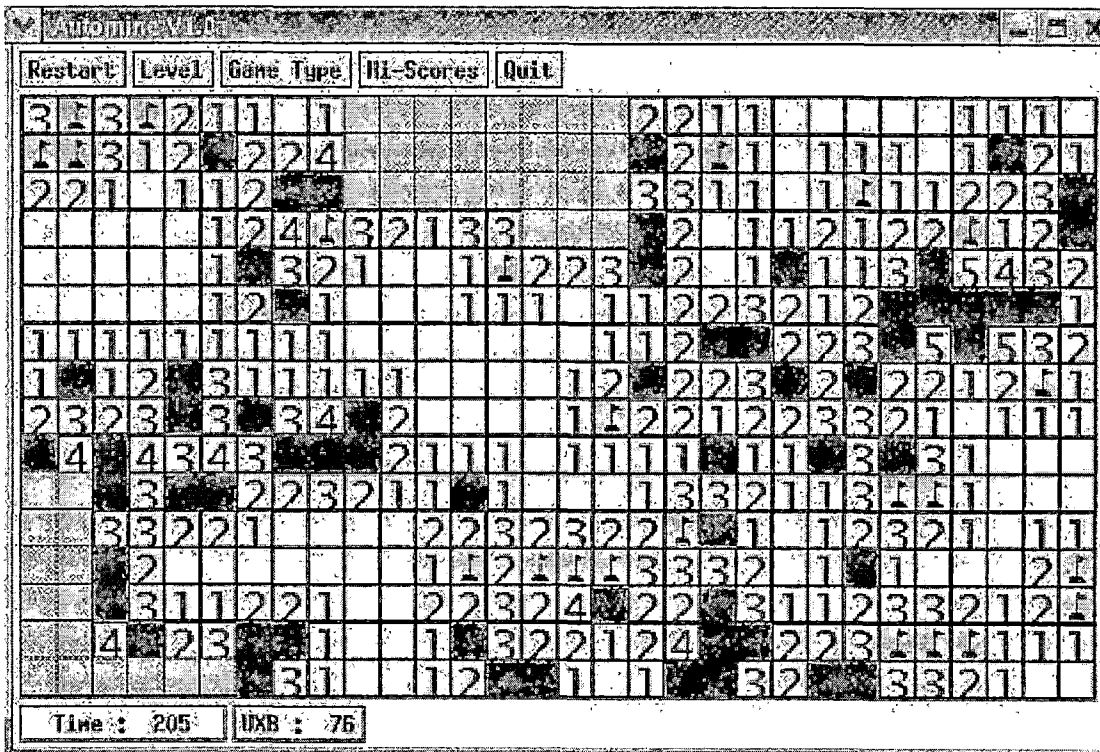


Figure 35. Need to Guess

Follow these procedures and there would be only red squares left as shown in Figure 37. Then mark all the red squares to win the game.

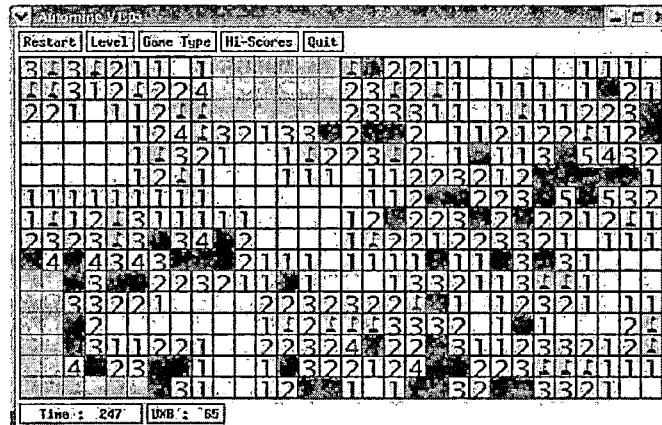


Figure 36. Get More Greens

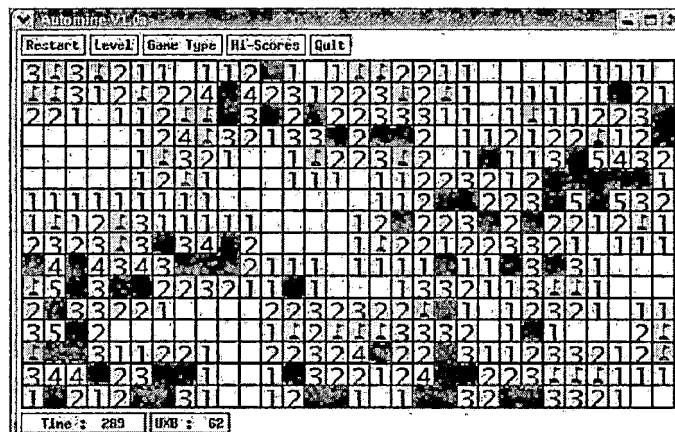


Figure 37. Only Red Squares Left

It is really difficult to finish the expert level without help. Here shows the game finished in 325 seconds with documentation process going on. The good results with automine would generate like 200 seconds performance according to the experiment with some careful users.

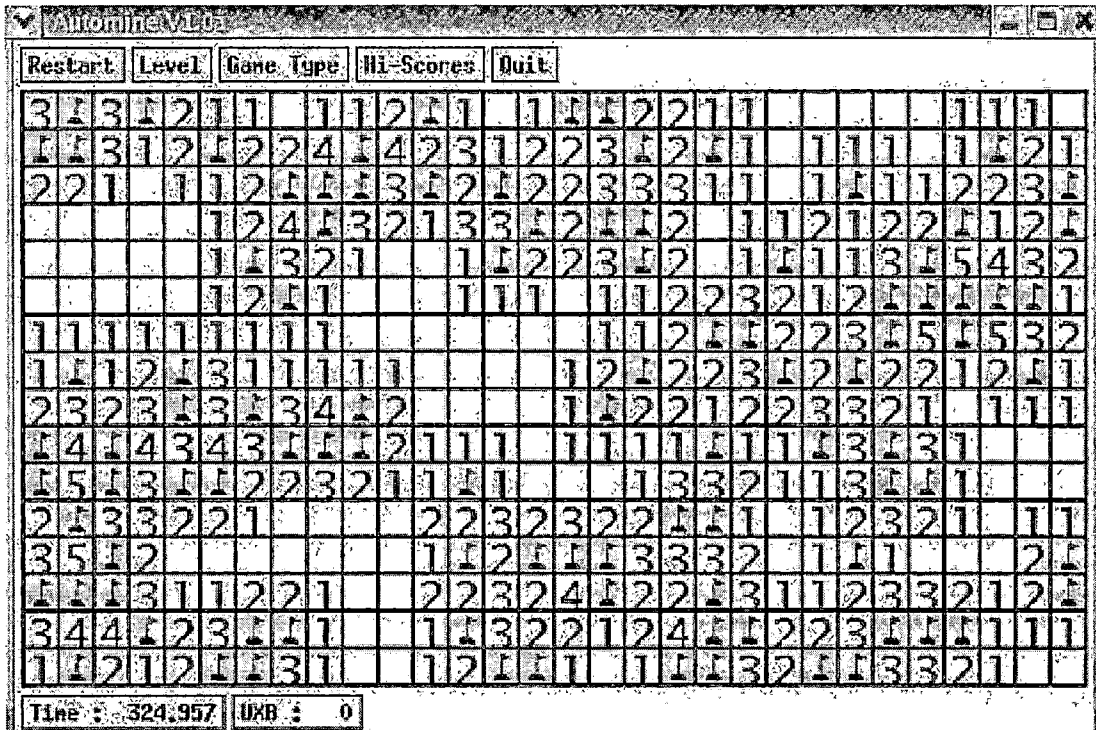


Figure 38. Game Won, Stop

Applying this procedure, sample users improve the time performance a lot. For beginner level, average improvement is 67% to become less than 10 sec. For intermediate and expert levels, sample users can't really finish one. But

the automine can help to achieve around 90 sec and 150 sec respectively.

Figure 39 shows the highest score of the beginner level. If you get the top ten score, it would show your latest score in dark. Otherwise, it would show your last score in the last row.

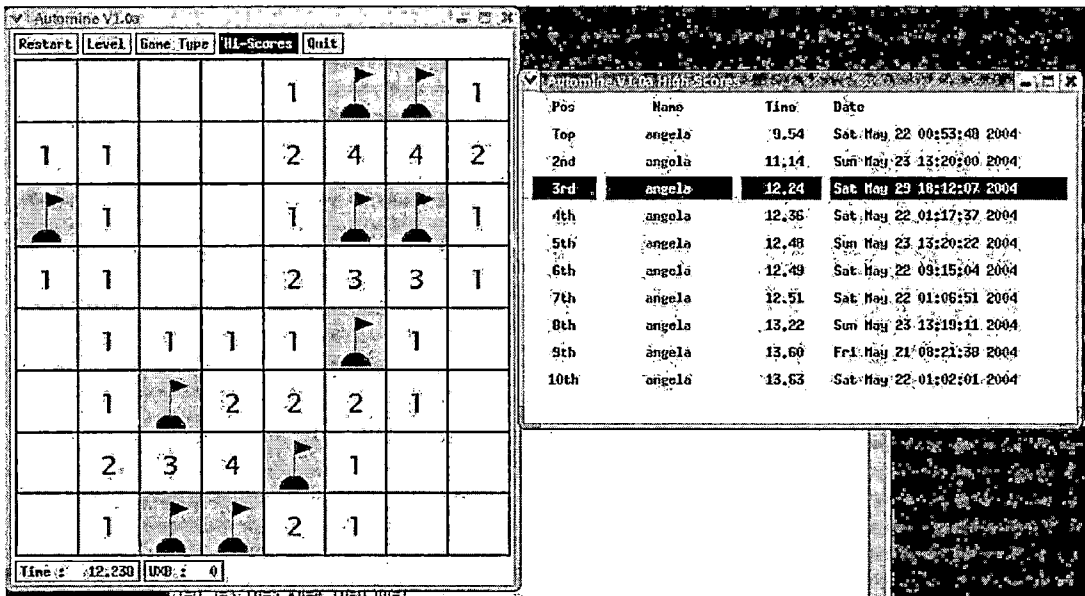


Figure 39. The Highest Score of Beginner Level

5.3 Pseudo Code

This section would illustrate the detailed implementation of automine algorithms. In the xbomb, there is a state variable showing the state of the square:

Table 5. State Variable of Xbomb

state	status
0-8	known, n=state
64-72	covered, n=state-64
80	Bomb; not used
112	think bomb

```
void FindRisk(void)
{
    First_Run{
        //update the average risk of the squares that have no
neighborhood information.
        if(ro nopt equal 0 and ro not equal
1)risk[i][j]equal ro; // far risk
        // risk: check valid number of neighbors
        // then compare with the known number with safe
        // and think_bomb numbers. Suitable for all cases
```



```

// including edges and corners.
    for(i=0;i<grid_width;i++)
        for(j=0;j<grid_height;j++)
            if (state[i][j]<Nbr) //uncovered
            {
                tb=0;kn=0;tn=0;un=0;
                for(k=-1;k<2;k++)
                    for(l=-1;l<2;l++)
                        {if(i+k>=0&&i+k<grid_width&&
j+1>=0&&j+1<grid_height) // avoid edge and corner
                            if(k!=0 or l!=0)
                                {tn++; //total neighbors
                                    if (state[i+k][j+1]<Nbr) kn++; //known
uncovered
                                        if (state[i+k][j+1]==Think_bomb) tb++;
//thinkbomb
                                    un=tn-kn-tb; //unknown
                                    if (un>0) //check the risk
                                    {
                                        if (tb==state[i][j]) r=0;//others must be
safe
                                            else if (state[i][j]-tb==un) r=1;
//others must be bombs
                                                else r=(double)(state[i][j]-tb)/un;
//assign equal Risk

```

```

        // update risk value
        for(k=-1;k<2;k++)
            for(l=-1;l<2;l++)
                {if(i+k>=0&&i+k<grid_width&&
j+1>=0&&j+1<grid_height)
                    if(k!=0 or l!=0)
                        {
                            if (state[i+k][j+1]>Nbr &&
state[i+k][j+1]!=Think_bomb) //covered
                                if (r==0 or risk[i+k][j+1]==0)
risk[i+k][j+1]=0;
                                else if (r==1 or risk[i+k][j+1]==1)
risk[i+k][j+1]=1;
                                    //else if
(risk[i+k][j+1]!=0||risk[i+k][j+1]!=1) //get the latest
                                        //else if
(risk[i+k][j+1]!=0&&risk[i+k][j+1]<r) //get the max Risk
                                            // risk[i+k][j+1]=r;
                                                else if
(risk[i+k][j+1]!=0||risk[i+k][j+1]!=1) //get f(a,b)=f(b,a)
                                                    risk[i+k][j+1]=(r+risk[i+k][j+1])/2.0;
//average
                else ;
            }

```

```

        } //for
Print information;
        }//uncovered left
    } //find risk
} // First_run
// Second Run Repeated to check all information used
do{
    keep record;
    update average risk;
    check risk;
    update risk;
    check if any risk value changed;
} //do
while (change!=0);
// Overlap
int ov=0;
    for(i=0;i<grid_width;i++)
        for(j=0;j<grid_height;j++)
            if (risk[i] [j]>0.1 and risk[i][j]<0.9)
                {ov=0;
                    for(k=-1;k<2;k++)
                        for(l=-1;l<2;l++)
                            {if(i+k>=0 and i+k<grid_width and
j+1>=0&&j+1<grid_height)

```

```

        if(k!=0 or l!=0)
            if (state[i+k][j+1]<Nbr) ov++;
    }

    if (ov==1) {risk[i][j]*=Add;}
    else if (ov>1) {risk[i][j]*=Reduce;}
}

// isolated
int iso=0,k1,l1,n1,m;
//0:12, 1:13, 2:14, 3:23, 4:24, 5:25, 6:34, 7:35, 8:45
static double r1[9][3]=
{0.04,0.21,0.37,    0.01,0.24,0.68,    0.00,0.25,1.0,
 0.09,0.43,0.42,    0.03,0.47,0.70,    0.00,0.50,1.0,
 0.13,0.65,0.46,    0.09,0.69,0.75,    0.27,0.79,0.61};
    for(i=1;i<grid_width-1;i++)
        for(j=1;j<grid_height-1;j++)
            {
                if(state[i][j]==1)
                    {
                        iso=0;
                        for(k=-1;k<2;k++)
                            for(l=-1;l<2;l++)
                                if(k!=0 or l!=0)
                                    {

```

```

        if(state[i+k][j+1]==2 and iso==0)
{iso=1;k1=k;l1=1;n1=0;}
        else if(state[i+k][j+1]==3 and iso==0)
{iso=1;k1=k;l1=1;n1=1;}
        else if(state[i+k][j+1]==4 and iso==0)
{iso=1;k1=k;l1=1;n1=2;}
        else
if(state[i+k][j+1]>63&&state[i+k][j+1]!=112){;}
        else {iso=2;break;}
    }
} //1
repeat for case 2, 3 and 4;
if(iso==1) // Yes isolation
    update risk;
} //for iso
// Draw Risk
for(i=0;i<grid_width;i++)
    for(j=0;j<grid_height;j++)
        if(state[i][j]>Nbr and
state[i][j]!=Think_bomb)
        {
            DrawRisk(i,j,risk[i][j]);
        } //draw
} // FindRisk

```

```
void DrawRisk(int x,int y,double value)
{
    /* Find the position. */
    // 0: black 1: red 2: yellow 3: blue 4: grey 5: green
6: white

    s=sqrt(value); //calculate side from risk
    if(value==0) draw green;
    else draw gray; then draw yellow/red;
}
```

CHAPTER SIX
MAINTENANCE MANUAL

6.1 Source Files

In the automine project, there are 13 source code files.

Table 6. Source Files

automine/COPYING	copyright
automine/FILES	Files Description
automine/LSM	Program Description
automine/README	Readme
automine/ChangeLog	Tracking changes
automine/Makefile	Compile
automine/hiscore.c	High score
automine/automine.c	Main program
automine/automine.h	Including header
automine/xwindow.c	Xwindow function
automine/automine.6	manual
automine/automine.ad	Application default
automine/icon.h	Icon

6.2 Installation Description

The installation of Automine is very simple. Just follow the two steps.

1. Copy the above files in 6.1 to an automine directory.
2. Run the makefile `$ make` (see APPENDIX C).

CHAPTER SEVEN

CONCLUSIONS AND FUTURE DIRECTION

7.1 Conclusions

In the automine project, the goal is to help users to achieve minimal time performance easily. The user should first randomly click several times to make the uncovered area as large as possible, more than 50%. It could hit and bomb but that is critical to achieve the minimal time. After there is a large uncovered area, automine would show there colors in hidden squares: green means absolutely safe, red means a bomb, yellow means unsure condition with the area proportional to the probability of risk. In many case, just follow the sequence to uncover or mark the closest green or red squares, the more information would lead to more certain decisions. However, there are still some cases that users need to guess when no green or red squares available. Users are suggested to choose the smallest risk. Keep in mind that sometimes the smallest risk one would still have a bomb hidden there.

The project covers the three typical levels: beginner (8x8, 10 bombs), intermediate (16x16, 40 bombs) and difficult (16x32, 80 bombs).

Three algorithms are applied:

1. Full search: to find out the green and red ones according to the known information.
2. Adjustment of overlap probability: When the hidden square has two or more neighbors uncovered, the risk probability calculated with the individual information would be adjusted. For example, the overlap one would take the maximum risk and the others should be reduced.
3. Certain Patterns: There are certain patterns whose probability could be calculated or simulated. With a table look up scheme, the probability of the neighbor squares could be identified precisely.

7.2 Future Direction

The most interesting automine project should be extended with a PC camera that can capture the information on any system and any variation of program. Then reproduce the information in the server computer, apply the automine algorithm and show suggestions on server computer.

Therefore, the user can use the help from the automine to beat any record in any machine.

The application should include the PC camera software development kit (SDK) and simple pattern recognition

algorithms to tell the numbers, safe area (0 bomb) and the marked bomb. These are the only information needed by the automine to provide the suggestions.

Besides, the xbomb has three types of minesweeper games: regular, triangle and hexagon. The algorithm could be applied to those special games with modification. Furthermore, the parallel processing algorithm using Spider System or PVM can be applied. The server will divide the problem into several segments and distribute to clients. One possible way is, for example, to segment the 8x8 field to four 5x5 fields. The other way is assign a new possible way to a new client to follow up.

APPENDIX A
SOURCE CODE OF AUTOMINE

```

//+++++
// Calculate Risk
//
// initialization
//+++++
void FindRisk(void)
{
int i,j,k,l,tb=0,kn=0,tn=0,un=0,change;
double r=0,ro=0,rol; //ro is the basic risk=covred bomb/covred square
static int Think_Bomb=112, Wrong_Bomb=98, Max_Nbr=8, Hidden=64;
static double Overlap_Add=1.05, Overlap_Reduce=0.9;
tb=0;kn=0;tn=0;un=0;
// First Run
for(i=0;i<grid_width;i++)
for(j=0;j<grid_height;j++)
{if(state[i][j]==Think_Bomb||state[i][j]==Wrong_Bomb) tb++;
if(state[i][j]<=Max_Nbr) kn++;
}
if(grid_height*grid_width-kn-tb>0)
ro=(double)(grid_bombs-tb)/(grid_height*grid_width-kn-tb);
for(i=0;i<grid_width;i++)
for(j=0;j<grid_height;j++)
if(ro!=0&&ro!=1)risk[i][j]=ro; // far risk

// find risk
for(i=0;i<grid_width;i++)
for(j=0;j<grid_height;j++)
if (state[i][j]<=Max_Nbr) //uncovered
{ tb=0;kn=0;tn=0;un=0;
for(k=-1;k<2;k++)
for(l=-1;l<2;l++)
{if(i+k>=0&&i+k<grid_width&& j+l>=0&&j+l<grid_height)
if(k!=0||l!=0)
{tn++; //total neighbors
if (state[i+k][j+l]<=Max_Nbr) kn++; //known uncovered
if (state[i+k][j+l]==Think_Bomb||state[i+k][j+l]==Wrong_Bomb)
tb++; //thinkbomb
}
}
un=tn-kn-tb; //unknown
if (un>0) //check the risk
{
if (tb==state[i][j]) r=0; //others must be safe
else if (state[i][j]-tb==un) r=1; //others must be bomb
else r=(double)(state[i][j]-tb)/un; //assign equal Risk

for(k=-1;k<2;k++)
for(l=-1;l<2;l++)
{if(i+k>=0&&i+k<grid_width&& j+l>=0&&j+l<grid_height)
if(k!=0||l!=0)
{
if
(state[i+k][j+l]>Max_Nbr&&state[i+k][j+l]!=Think_Bomb&&state[i+k][j+l]!=Wrong_Bomb) //covered
if (r==0||risk[i+k][j+l]==0) risk[i+k][j+l]=0;
else if (r==1||risk[i+k][j+l]==1) risk[i+k][j+l]=1;
//else if (risk[i+k][j+l]!=0||risk[i+k][j+l]!=1) //get the latest
//else if (risk[i+k][j+l]!=0&&risk[i+k][j+l]<r) //get the max
Risk
// risk[i+k][j+l]=r;
else if (risk[i+k][j+l]!=0||risk[i+k][j+l]!=1) //get f(a,b)=f(b,a)
risk[i+k][j+l]=(r+risk[i+k][j+l])/2.0; //average
else ;
}
}
}
}
}
}
}
}

```

```

    }
    } //for
fprintf(stderr, "i=%d j=%d tn=%d kn=%d tb=%d un=%d\n", i, j, tn, kn, tb, un);
    } //uncovered left
    } //find risk

// Second Run
change=0;
do{
    for(i=0;i<grid_width;i++)
        for(j=0;j<grid_height;j++)
            risk1[i][j]=risk[i][j]; //keep record
    rol=ro;
    tb=0;kn=0;tn=0;un=0;
    for(i=0;i<grid_width;i++)
        for(j=0;j<grid_height;j++)
            {if(state[i][j]==Think_Bomb||state[i][j]==Wrong_Bomb) tb++;
             else if (risk[i][j]==1) tb++;
             else if (state[i][j]<=Max_Nbr) kn++;
             else if (risk[i][j]==0) kn++;
             else ;
            }
        if(grid_height*grid_width-kn-tb>0)
            ro=(double)(grid_bombs-tb)/(grid_height*grid_width-kn-tb);

    for(i=0;i<grid_width;i++)
        for(j=0;j<grid_height;j++)
            if (risk[i][j]==rol&&ro!=0&&ro!=1) risk[i][j]=ro; // update far risk

// update risk
    for(i=0;i<grid_width;i++)
        for(j=0;j<grid_height;j++)
            if (state[i][j]<=Max_Nbr) //uncovered
                { tb=0;kn=0;tn=0;un=0;
                 for(k=-1;k<2;k++)
                     for(l=-1;l<2;l++)
                         {if(i+k>=0&&i+k<grid_width&& j+l>=0&&j+l<grid_height)
                          if(k!=0||l!=0)
                              {tn++; //total neighbors
                              //consider risk==0 or 1
                              if (state[i+k][j+l]<=Max_Nbr) kn++;
                              else if (risk[i+k][j+l]==0) kn++; //known uncovered
                              else if (state[i+k][j+l]==Think_Bomb||state[i+k][j+l]==Wrong_Bomb)
                                  tb++;
                              else if (risk[i+k][j+l]==1) tb++; //thinkbomb
                              else ;
                              }
                          }
                 un=tn-kn-tb; //unknown
                 if (un>0) //check the risk
                     {
                         if (tb==state[i][j]) r=0;//others must be safe
                         else if (state[i][j]-tb==un) r=1; //others must be bomb
                         else r=(double)(state[i][j]-tb)/un; //assign equal Risk

                         for(k=-1;k<2;k++)
                             for(l=-1;l<2;l++)
                                 {if(i+k>=0&&i+k<grid_width&& j+l>=0&&j+l<grid_height)
                                  if(k!=0||l!=0)
                                      {
                                          if
                                          (state[i+k][j+l]>Max_Nbr&&state[i+k][j+l]!=Think_Bomb&&state[i+k][j+l]!=Wrong_Bomb) //covered

```

```

        if (risk[i+k][j+1]!=0&&risk[i+k][j+1]!=1) //covered
        if (r==0||risk[i+k][j+1]==0) risk[i+k][j+1]=0;
        else if (r==1||risk[i+k][j+1]==1) risk[i+k][j+1]=1;
        //else if (risk[i+k][j+1]!=0||risk[i+k][j+1]!=1) //get the latest
        //else if (risk[i+k][j+1]!=0&&risk[i+k][j+1]<r) //get the max

Risk
        // risk[i+k][j+1]=r;
        else if (risk[i+k][j+1]<=r)//get f(a,b)=f(b,a)
            risk[i+k][j+1]=r;
            //risk[i+k][j+1]=(r+risk[i+k][j+1])/2.0; //average
    }
    } //for
} //uncovered left
} //find risk
change=0;
for(i=0;i<grid_width;i++)
    for(j=0;j<grid_height;j++)
        if(risk[i][j]>=0&&risk[i][j]<=1)
            if((int)(100*risk1[i][j])!=(int)(100*risk[i][j])) {change=1;
//fprintf(stderr, "*****\n%d %d %f %f\n",i,j,risk1[i][j],risk[i][j]);
                break;}
} //do
while (change!=0);

// Overlap
int ov=0;
for(i=0;i<grid_width;i++)
    for(j=0;j<grid_height;j++)
        if(risk[i][j]>0.1 && risk[i][j]<0.9)
        {ov=0;
        for(k=-1;k<2;k++)
            for(l=-1;l<2;l++)
                {if(i+k>=0&&i+k<grid_width&& j+l>=0&&j+l<grid_height)
                if(k!=0||l!=0)
                    if (state[i+k][j+l]<=Max_Nbr) ov++;
                }
            if (ov==1) {risk[i][j]*=Overlap_Reduce;}
            else if (ov>1) {risk[i][j]*=Overlap_Add;}
        }

// isolated
int iso=0,k1,l1,n1,m;
//0:12, 1:13, 2:14, 3:23, 4:24, 5:25, 6:34, 7:35, 8:45
static double r1[9][3]={0.04,0.21,0.37, 0.01,0.24,0.68, 0.00,0.25,1.0,
                        0.09,0.43,0.42, 0.03,0.47,0.70, 0.00,0.50,1.0,
                        0.13,0.65,0.46, 0.09,0.69,0.75, 0.27,0.79,0.61};
for(i=1;i<grid_width-1;i++)
    for(j=1;j<grid_height-1;j++)
    {
        iso=0;
        if(state[i][j]==1)
        {
            iso=0;
            for(k=-1;k<2;k++)
                for(l=-1;l<2;l++)
                    if(k!=0||l!=0)
                    {
                        if(state[i+k][j+l]==2&&iso==0) {iso=1;k1=k;l1=l;n1=0;}
                        else if(state[i+k][j+l]==3&&iso==0) {iso=1;k1=k;l1=l;n1=1;}
                        else if(state[i+k][j+l]==4&&iso==0) {iso=1;k1=k;l1=l;n1=2;}
                        else
                            if(state[i+k][j+l]>=Hidden&&state[i+k][j+l]!=Think_Bomb&&state[i+k][j+l]!=Wrong_Bomb){;}
                    }
        }
    }

```

```

        else {iso=2;break;}
    }
} //1
else if(state[i][j]==2)
{
iso=0;
for(k=-1;k<2;k++)
for(l=-1;l<2;l++)
if(k!=0||l!=0)
{
if(state[i+k][j+1]==3&&iso==0) {iso=1;k1=k;l1=l;n1=3;}
else if(state[i+k][j+1]==4&&iso==0) {iso=1;k1=k;l1=l;n1=4;}
else if(state[i+k][j+1]==5&&iso==0) {iso=1;k1=k;l1=l;n1=5;}
else
if(state[i+k][j+1]>=Hidden&&state[i+k][j+1]!=Think_Bomb&&state[i+k][j+1]!=Wrong
_Bomb){;}
        else {iso=2;break;}
    }
} //2
else if(state[i][j]==3)
{
iso=0;
for(k=-1;k<2;k++)
for(l=-1;l<2;l++)
if(k!=0||l!=0)
{
if(state[i+k][j+1]==4&&iso==0) {iso=1;k1=k;l1=l;n1=6;}
if(state[i+k][j+1]==5&&iso==0) {iso=1;k1=k;l1=l;n1=7;}
else
if(state[i+k][j+1]>=Hidden&&state[i+k][j+1]!=Think_Bomb&&state[i+k][j+1]!=Wrong
_Bomb){;}
        else {iso=2;break;}
    }
} //3
else if(state[i][j]==4)
{
iso=0;
for(k=-1;k<2;k++)
for(l=-1;l<2;l++)
if(k!=0||l!=0)
{
if(state[i+k][j+1]==5&&iso==0) {iso=1;k1=k;l1=l;n1=8;}
else
if(state[i+k][j+1]>=Hidden&&state[i+k][j+1]!=Think_Bomb&&state[i+k][j+1]!=Wrong
_Bomb){;}
        else {iso=2;break;}
    }
} //4

if(iso==1) // Yes isolation
{ //update
if((k1==1||k1==-1)&&l1==0)
{
for(m=-1;m<2;m++) {n01(&risk[i+2*k1][j+m],r1[n1][2]);
n01(&risk[i+k1][j+m],r1[n1][1]);
n01(&risk[i][j+m],r1[n1][1]);
n01(&risk[i-k1][j+m],r1[n1][0]);}
}
if((l1==1||l1==-1)&&k1==0)
{
for(m=-1;m<2;m++) {n01(&risk[i+m][j+2*l1],r1[n1][2]);
n01(&risk[i+m][j+l1],r1[n1][1]);
n01(&risk[i+m][j],r1[n1][1]);}
}
}
}

```



```

        n01(&risk[i+m][j-1],r1[n1][0]);
    }
} //update
} //for iso

// Draw Risk
for(i=0;i<grid_width;i++)
    for(j=0;j<grid_height;j++)
        if(state[i][j]>Max_Nbr && state[i][j]!=Think_Bomb&&
state[i][j]!=Wrong_Bomb)
        {
            DrawRisk(i,j,risk[i][j]);
        } //draw
} // FindRisk

// check if the old risk is neither 0 nor 1 then update with new risk
void n01(double *m, double v)
{
    if (*m>0&&*m<1) *m=v;
}

```

APPENDIX B
SOURCE CODE OF SIMULATION

B.1 SOURCE CODE

```
// Automine Minesweeper Utility
// Test the probability of special pattern
// Angela Tzujui Huang
// Beta Version 1.0
// 10/04/2002

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

//global variables
//mine number; max 50x50
#define Max 50
#define Nbr 8 //Neighbor
#define Bomb 9
#define Sim_Times 1E4
#define BEGINER 1
#ifdef BEGINER
#define ROW 8
#define COL 8
#define Pmn 10
#endif
#ifdef INTERMEDIATE
#define ROW 16
#define COL 16
#define Pmn 40
#endif
#ifdef EXPERT
#define ROW 16
#define COL 32
#define Pmn 100
#endif
int board[Max][Max], hit[Max][Max];
//Define Pattern
int PRow=4, PCol=3;
int N1=2,N2=3;
int valid=0;
int Row=ROW, Col=COL, pmn=Pmn;

int sum(int i, int j)
{
    int s;
    if(i==0&&j==0) s=(board[i][j+1]>Nbr)+(board[i+1][j]>Nbr)+(board[i+1][j+1]>Nbr);
    else if(i==Row-1&&j==0) s=(board[i][j+1]>Nbr)+(board[i-1][j]>Nbr)+(board[i-1][j+1]>Nbr);
    else if(i==0&&j==Col-1) s=(board[i][j-1]>Nbr)+(board[i+1][j]>Nbr)+(board[i+1][j-1]>Nbr);
    else if(i==Row-1&&j==Col-1) s=(board[i][j-1]>Nbr)+(board[i-1][j]>Nbr)+(board[i-1][j-1]>Nbr);

    else if(i==0) s=(board[i+1][j-1]>Nbr)+(board[i+1][j]>Nbr)+(board[i+1][j+1]>Nbr)
        +(board[i][j-1]>Nbr)+(board[i][j+1]>Nbr);
    else if(i==Row-1) s=(board[i-1][j-1]>Nbr)+(board[i-1][j]>Nbr)+(board[i-1][j+1]>Nbr)
        +(board[i][j-1]>Nbr)+(board[i][j+1]>Nbr);
    else if(j==0) s=(board[i-1][j+1]>Nbr)+(board[i-1][j+1]>Nbr)+(board[i+1][j+1]>Nbr)
        +(board[i-1][j]>Nbr)+(board[i+1][j]>Nbr);
    else if(j==Col-1) s=(board[i-1][j+1]>Nbr)+(board[i][j+1]>Nbr)+(board[i+1][j+1]>Nbr)
        +(board[i-1][j]>Nbr)+(board[i+1][j]>Nbr);

    else { s=(board[i-1][j-1]>Nbr)+(board[i-1][j]>Nbr)+(board[i-1][j+1]>Nbr)
        +(board[i][j-1]>Nbr)+(board[i][j+1]>Nbr)
        +(board[i+1][j-1]>Nbr)+(board[i+1][j]>Nbr)+(board[i+1][j+1]>Nbr);
        //printf("i=%d %d %d %d %d\n",i,j, (int)(board[i][j-1]>Nbr),board[i][j+1]>Nbr,s);
    }
}

return s;
}

void new_board()
```

```

{
    int n=Row*Col,tmp[Max*Max];
    int i,j,k,r,c;

    for(i=0;i<n;i++) tmp[i]=i;
    for(i=0;i<pmm;i++)
    {
        k=drand48()*n;
        r=tmp[k]/Col;
        c=tmp[k]%Col;
        board[r][c]=Bomb;
        tmp[k]=tmp[n-1];
        n--;
    }

    for(i=0;i<Row;i++)
    for(j=0;j<Col;j++)
    {
        if (board[i][j]==0)
            board[i][j]=sum(i,j);
    }

//Print board
/*
    for(i=0;i<Row;i++)
    {
        for(j=0;j<Col;j++)
            printf("%d ",board[i][j]);
        printf("\n");
    }
*/
}

void stat()
{
    int i,j,k,r,c,x,y;

    for(i=2;i<Row-2;i++)
    for(j=2;j<Col-2;j++)
    {
        if(board[i][j]==N1)
        {
            if(board[i+1][j]==N2)
            { for(r=0;r<PRow;r++) for(c=0;c<PCol;c++)
                hit[r][c]+=(board[i+r-1][j+c-1]>Nbr);
                valid++;}
            if(board[i-1][j]==N2)
            { for(r=0;r<PRow;r++) for(c=0;c<PCol;c++)
                hit[r][c]+=(board[i-r+1][j-c+1]>Nbr);
                valid++;}
            if(board[i][j+1]==N2)
            { for(r=0;r<PRow;r++) for(c=0;c<PCol;c++)
                hit[r][c]+=(board[i-c+1][j+r-1]>Nbr);
                valid++;}
            if(board[i][j-1]==N2)
            { for(r=0;r<PRow;r++) for(c=0;c<PCol;c++)
                hit[r][c]+=(board[i+c-1][j-r+1]>Nbr);
                valid++;}
        }
    }
}

main()
{
    int i,j,k,r,c;
    float avg;

// random seed

```

```

printf("%d\n",i=(int)time());
srand48(i);

scanf("%d %d %d %d %d",&Row, &Col, &pmn, &N1, &N2);

// initialization
for(i=0;i<PRow;i++)
for(j=0;j<PCol;j++)
hit[i][j]=0;

// Begin

for(k=0;k<Sim_Times;k++)
{
for(i=0;i<Row;i++)
for(j=0;j<Col;j++)
{
board[i][j]=0;
}
new_board();
stat();
}

printf("\nReport.... Row= %d Col= %d pmn= %d\n\n",Row, Col, pmn);

// Report

printf("Valid sample: %d\n\n",valid);

hit[1][1]=N1*valid;
hit[2][1]=N2*valid;
for(i=0;i<PRow;i++)
{
for(j=0;j<PCol;j++)
printf("%f ",(float)hit[i][j]/valid);

printf("\n");
}
printf("\n");
avg=0;
for(i=0;i<PRow;i++)
for(j=0;j<PCol;j++)
avg+=hit[i][j];
printf("avg= %f\n\n",(float)avg/valid-N1-N2);
}

```

B.2 RUN SCRIPT

```

echo "8 8 10 1 1" | ./sim_mine>>t.dat
echo "8 8 10 1 2" | ./sim_mine>>t.dat
echo "8 8 10 1 3" | ./sim_mine>>t.dat
echo "8 8 10 1 4" | ./sim_mine>>t.dat
echo "8 8 10 2 2" | ./sim_mine>>t.dat
echo "8 8 10 2 3" | ./sim_mine>>t.dat
echo "8 8 10 2 4" | ./sim_mine>>t.dat
echo "8 8 10 3 3" | ./sim_mine>>t.dat
echo "8 8 10 3 4" | ./sim_mine>>t.dat

echo "16 16 40 1 1" | ./sim_mine>>t.dat
echo "16 16 40 1 2" | ./sim_mine>>t.dat
echo "16 16 40 1 3" | ./sim_mine>>t.dat
echo "16 16 40 1 4" | ./sim_mine>>t.dat
echo "16 16 40 2 2" | ./sim_mine>>t.dat
echo "16 16 40 2 3" | ./sim_mine>>t.dat
echo "16 16 40 2 4" | ./sim_mine>>t.dat
echo "16 16 40 3 3" | ./sim_mine>>t.dat
echo "16 16 40 3 4" | ./sim_mine>>t.dat

echo "16 32 99 1 1" | ./sim_mine>>t.dat
echo "16 32 99 1 2" | ./sim_mine>>t.dat

```

```

16
echo "16 32 99 1 3" | ./sim_mine>>t.dat
echo "16 32 99 1 4" | ./sim_mine>>t.dat
echo "16 32 99 1 5" | ./sim_mine>>t.dat
echo "16 32 99 2 2" | ./sim_mine>>t.dat
echo "16 32 99 2 3" | ./sim_mine>>t.dat
echo "16 32 99 2 4" | ./sim_mine>>t.dat
echo "16 32 99 2 5" | ./sim_mine>>t.dat
echo "16 32 99 3 3" | ./sim_mine>>t.dat
echo "16 32 99 3 4" | ./sim_mine>>t.dat
echo "16 32 99 3 5" | ./sim_mine>>t.dat
echo "16 32 99 4 4" | ./sim_mine>>t.dat
echo "16 32 99 4 5" | ./sim_mine>>t.dat

```

B.3 RESULT

1086073538

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 90510

```

0.106927 0.105038 0.103944
0.171141 1.000000 0.171583
0.170688 1.000000 0.170677
0.104331 0.105215 0.106364

```

avg= 1.315910

1086073538

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 43272

```

0.038200 0.038709 0.038801
0.221714 1.000000 0.222037
0.220258 2.000000 0.220281
0.370933 0.376826 0.367952

```

avg= 2.115710

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 6899

```

0.011886 0.011016 0.014350
0.234962 1.000000 0.239600
0.242354 3.000000 0.245833
0.668503 0.693724 0.675025

```

avg= 3.037252

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 384

```

0.000000 0.000000 0.000000
0.252604 1.000000 0.250000
0.242188 4.000000 0.255208
1.000000 1.000000 1.000000

```

avg= 4.000000

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 40377

0.181985 0.180623 0.178914
0.366372 2.000000 0.362682
0.362211 2.000000 0.367214
0.177453 0.180573 0.183496

avg= 2.541521

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 12923

0.082798 0.077923 0.079161
0.438753 2.000000 0.435735
0.443473 3.000000 0.442157
0.414300 0.419407 0.406175

avg= 3.239882

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 1775

0.027042 0.024225 0.029859
0.487887 2.000000 0.490141
0.458028 4.000000 0.482817
0.710423 0.689577 0.681127

avg= 4.081127

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 7113

0.242654 0.250387 0.248840
0.568536 3.000000 0.566428
0.561226 3.000000 0.561929
0.249684 0.246169 0.246028

avg= 3.741881

1086073539

Report.... Row= 8 Col= 8 pmn= 10

Valid sample: 1657

0.143030 0.123718 0.120097
0.679541 3.000000 0.665661
0.638503 4.000000 0.629451
0.476162 0.441762 0.468920

avg= 4.386844

1086073539

Report.... Row= 16 Col= 16 pmn= 40

Valid sample: 795102

0.099985 0.099509 0.099236
0.175145 1.000000 0.175327

0.175335 1.000000 0.175463
0.099263 0.099479 0.099988

avg= 1.298730

1086073540

Report.... Row= 16 Col= 16 pnn= 40

Valid sample: 359524

0.040790 0.040220 0.040278
0.218428 1.000000 0.218353
0.221051 2.000000 0.220881
0.371594 0.375978 0.373716

avg= 2.121288

1086073540

Report.... Row= 16 Col= 16 pnn= 40

Valid sample: 60339

0.014667 0.014153 0.013706
0.238585 1.000000 0.241784
0.239033 3.000000 0.238072
0.678649 0.679710 0.684168

avg= 3.042526

1086073541

Report.... Row= 16 Col= 16 pnn= 40

Valid sample: 3364

0.000000 0.000000 0.000000
0.244946 1.000000 0.252973
0.240785 4.000000 0.261296
1.000000 1.000000 1.000000

avg= 4.000000

1086073541

Report.... Row= 16 Col= 16 pnn= 40

Valid sample: 341355

0.188575 0.186179 0.186158
0.359585 2.000000 0.359596
0.360088 2.000000 0.359819
0.186211 0.186495 0.188206

avg= 2.560912

1086073542

Report.... Row= 16 Col= 16 pnn= 40

Valid sample: 118816

0.089079 0.088700 0.089811
0.433157 2.000000 0.433081
0.433065 3.000000 0.433107
0.422746 0.422830 0.422014

avg= 3.267590

1086073542

Report.... Row= 16 Col= 16 pmm= 40

Valid sample: 17341

0.032812 0.033966 0.033677
0.473502 2.000000 0.474655
0.477135 4.000000 0.474252
0.694193 0.708091 0.698172

avg= 4.100456

1086073543

Report.... Row= 16 Col= 16 pmm= 40

Valid sample: 71520

0.269114 0.268820 0.266303
0.549692 3.000000 0.548923
0.548280 3.000000 0.548867
0.266848 0.269379 0.268009

avg= 3.804237

1086073543

Report.... Row= 16 Col= 16 pmm= 40

Valid sample: 18813

0.154893 0.153192 0.158188
0.629777 3.000000 0.636262
0.633870 4.000000 0.633817
0.492053 0.489024 0.485196

avg= 4.466273

1086073544

Report.... Row= 16 Col= 32 pmm= 99

Valid sample: 1505522

0.118099 0.117747 0.118533
0.161066 1.000000 0.161650
0.161789 1.000000 0.161116
0.118370 0.117798 0.118211

avg= 1.354379

1086073545

Report.... Row= 16 Col= 32 pmm= 99

Valid sample: 837394

0.050655 0.051064 0.050833
0.212022 1.000000 0.211751
0.211841 2.000000 0.211835
0.384150 0.384285 0.384117

avg= 2.152552

1086073546

Report.... Row= 16 Col= 32 pmm= 99

Valid sample: 180653

0.018411 0.018538 0.018234

0.236359 1.000000 0.235346
0.235922 3.000000 0.237190
0.685087 0.684821 0.685275

avg= 3.055183

1086073547

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 13315

0.000000 0.000000 0.000000
0.252572 1.000000 0.253323
0.249643 4.000000 0.244461
1.000000 1.000000 1.000000

avg= 4.000000

1086073548

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 0

nan nan nan
nan nan nan
nan nan nan
nan nan nan

avg= nan

1086073549

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 901694

0.213000 0.213038 0.212986
0.340433 2.000000 0.340100
0.340339 2.000000 0.340104
0.212916 0.212928 0.213180

avg= 2.639024

1086073550

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 377893

0.110486 0.110314 0.110275
0.417605 2.000000 0.417658
0.416515 3.000000 0.417147
0.443686 0.444173 0.443215

avg= 3.331075

1086073551

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 70280

0.045120 0.046429 0.044863
0.468284 2.000000 0.465581
0.465894 4.000000 0.463830
0.713389 0.710031 0.712991

avg= 4.136411

1086073552

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 4542

0.000000	0.000000	0.000000
0.498239	2.000000	0.498899
0.503963	5.000000	0.498899
1.000000	1.000000	1.000000

avg= 5.000000

1086073553

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 267376

0.297985	0.298142	0.299215
0.525556	3.000000	0.527157
0.527224	3.000000	0.524722
0.298965	0.298191	0.298187

avg= 3.895342

1086073554

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 85212

0.186453	0.183484	0.183530
0.612296	3.000000	0.612907
0.611991	4.000000	0.609339
0.515855	0.519094	0.518519

avg= 4.553467

1086073555

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 12637

0.087125	0.086729	0.084039
0.687426	3.000000	0.686318
0.684814	5.000000	0.683548
0.752315	0.751840	0.753739

avg= 5.257893

1086073556

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 42071

0.390293	0.386656	0.383756
0.711868	4.000000	0.707899
0.707708	4.000000	0.711820
0.383542	0.386394	0.390768

avg= 5.160705

1086073557

Report.... Row= 16 Col= 32 pmn= 99

Valid sample: 9850

0.269848 0.275330 0.278173
0.796650 4.000000 0.791878
0.793503 5.000000 0.794619
0.611574 0.608122 0.603655

avg= 5.823350

APPENDIX C

MAKEFILE

```

# This file Copyright 2004 Angela T. Huang
# It may be distributed under the GNU Public License, version 2, or
# any higher version. See section COPYING of the GNU Public license
# for conditions under which this file may be redistributed.
#

CC=gcc
CFLAGS=-O2

INCLUDES=

LIB=

XLIB=-L/usr/X11R6/lib -lXaw -lXmu -lXt -lX11

COMPILE=$(CC) -c $(CFLAGS)

LINK=$(CC)

OBJ=automine.o xwindow.o hiscore.o

INSTDIR=/usr/local

#####

automine : $(OBJ)
        $(LINK) $(OBJ) -o $@ $(LIB) $(XLIB)

#####

%.o : %.c
        $(COMPILE) $< -o $@ $(INCLUDES)

automine.o : automine.c automine.h
xwindow.o : xwindow.c automine.h icon.h
hiscore.o : hiscore.c automine.h

#####

clean :
        -rm -f *.o *~ core

#####

install :
        strip automine
        install -d $(INSTDIR)/bin
        install -d $(INSTDIR)/man/man6
        install -d $(INSTDIR)/lib/app-defaults
        install -m 755 automine $(INSTDIR)/bin
        install -m 644 automine.6 $(INSTDIR)/man/man6
        install -m 644 automine.ad $(INSTDIR)/lib/app-defaults/Automine

```

REFERENCES

- [1] Clay Mathematics Institute, "Million-Dollar Minesweeper,"
<http://www.claymath.org/prizeproblems/milliondollarminesweeper.htm>
- [2] IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1993).
- [3] Kevin Wright and Hudelson, "The Math of Minesweeper"
http://leibrand.net/kevin/mine_paper.htm
- [4] Minesweeper: Advanced Tactics
<http://www.nothings.org/games/minesweeper>
- [5] Minesweeper Analysis and Strategies
http://fvdp.homestead.com/files/msw_index.html
- [6] Richard Botting, "Minesweeper End Game"
<http://www.csci.csusb.edu/dick/cs320/prolog/mine.plg>
- [7] Richard Kaye, "Some Minesweeper Configurations,"
<http://www.mat.bham.ac.uk/R.W.Kaye>
- [8] The Minesweeper Page- About Probabilities
<http://www.frankwester.net>