PREDICTING THE NEXT MOVE: PATTERN GENERATION IN THE GAME OF GO

A PROJECT REPORT

Submitted by

ADITI SINGH (09BCE003)

In partial fulfilment for the award

Of

B. Tech

Degree in

Computer Science and Engineering

School of Computing Science and Engineering

VIT® UNIVERSITY (Estd. u/s 3 of UGC Act 1956) VELLORE ▪ CHENNAI www.vit.ac.in

May - 2013

School of Computing Science and Engineering

Course: *Project Work (CSE499)*

WINTER SEMESTER 2012-13

Faculty: Professor Kumar K.

Group Id: 1036

Name of the Student:	ADITI SINGH
Registration No:	09BCE003

Title of Project:

Generating Patterns from Go Game Records

Project Coordinator	**Program Chair**	**Dean**
B.Tech.CSE	**B.Tech.CSE**	**SCSE**

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ▪ CHENNAI
www.vit.ac.in

School of Computing Science and Engineering

DECLARATION

I hereby declare that the project entitled **“Predicting the Next Move: PATTERN GENERATION IN THE GAME OF GO.”** submitted by me to the School of Computing Science and Engineering, VIT University, Vellore-14 in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out by me/us under the supervision of **Professor Kumar K, Assistant Professor (Selection Grade).** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

ADITI SINGH (09BCE003)

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ■ CHENNAI
www.vit.ac.in

School of Computing Science and Engineering

CERTIFICATE

The project report entitled **“Predicting the Next Move: PATTERN GENERATION IN THE GAME OF GO.”** is prepared and submitted by **ADITI SINGH (Register No: 00BCE003).** It has been found satisfactory in terms of scope, quality and presentation as partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** in VIT University, India.

Professor Kumar K.

(Name & Signature of the Internal Guide)

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

I would like to express my profound gratitude to my Project Guide- Professor Kumar K (Assistant Professor, Selection Grade), SCSE, VIT University, who guided me throughout the project work and helped me out with my thesis. I would also like to thank my Programme chair, Professor Anand M. for his guidance throughout the project duration. Lastly, I would like to thank our Dean- Professor Margaret Announcia, for her dedicated efforts in making this whole procedure run smoothly.

Thank you.

Place: Vellore

Date: 14-05-2013.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Expansion
SGF	Smart Game Format

ABSTRACT

In this project, user behaviour models are designed using a data driven method. Previously, player models were designed using user surveys, small-scale observation experiments, or knowledge engineering. These methods, though producing a semantically meaningful model, were limited in their applicability. To address this limitation, a data-driven methodology for generating ‘player models’ based on past observations of other professional players is developed. The base assumption for using this methodology is that we can accurately predict what a player will do in a given situation (his next counter move, attack or defence), if we examine enough data from former players that were in similar situations.

The method of limiting the number of positions in which a player can make a move, is particularly important because we know that unlike the game of chess (which is similar since it is also a zero sum, two player board game), the search tree branches out indefinitely in the game of Go. The number of possible positions for a player to keep his stone is 3^381.

By using game records we could devise a lot of move patterns which are a perfect and a concise way of defining rules.

GNU-Go which generates patterns for go game has hand-made pattern database [3]. Though, most of the patterns are covered and nicely classified as Attack, KO, Eye etc. a better method had to be devised so that human errors which creep in, are minimized.

Thus a method to automatically generate the patterns by reading through the electronic versions of the professional game records in the SGF format was devised.

PROJECT DESCRIPTION:

- The generation of patterns from large game corpuses to represent the many rules of the game of go.
- Thus patterns are computer generated by reading a lot of game records. Opening GO board theory is evaluated, wherein the important theory of “JOSEKI” will be studied. These patterns are important since opening game moves will determine how the game will proceed further.
- Secondly, other rules via patterns would be extracted from the game records by first extracting only relevant data from the whole SGF file , storing it in the database, reading those moves and from the board state which these moves form, 5 by 5 and 3 by 3 arrays of pattern are formed.

- These patterns and their canonical forms by subsequent rotation and reflection then by flipping of colours are compared and taken as one. The frequency of the distinct patterns are recorded and stored in the pattern database
- The game of ‘Go’ is still a game where research is going on. This game has very simple set of rules but best ‘computer-go’ still fails to beat the best human player.

Thus I propose to devise methods which would make prediction easier, faster and more accurate. Opening game theory is important in learning the patterns, which are used frequently, to determine how the game proceeds further. Further other patterns are used to find move states which are most often used by professional go players.

1. INTRODUCTION:

1.1 General :

The ability to determine how players will respond to situations in game can be invaluable information both to game designers and to AI management systems trying to tailor games to players’ behaviour. In the past, researchers have created these models manually, however these techniques can be time-consuming to execute, prone to human error and result in models with limited generalizability, i.e. they are applicable only for a single particular game for which they are surveyed.

As players of different type progress through a game, they react to the situation they are in a different type of way. If we assume that similar types of players will select the same actions in a given situation we can use observations of former players as a basis for a model of future player behaviour. By observing the actions a player takes as they progress through the game, we can compare these to the actions that previous players have made in similar situations and then predict what the current player will do.

Thus the previous game records which are now available in the electronic versions, in this case the SGF format, which contains all the information available from the game place, date, player ranks to the final game moves in sequence, is used as a way to capture player behaviour.

1.2 Motivation:

Here, when we refer to player actions, we are primarily concerned with evidence of the game content player’s experience. Using a data-driven approach to player modelling, one would only need to observe the actions that a player takes in order to construct the overall model of player behaviour. Since our approach is dependent solely on observing the actions that a user takes, we only need an ordered list of the attributes that characterize a user's experience as she progresses through the game.

One of the main benefits of this data-driven approach is we are agnostic to player type and therefore do not rely on a semantically meaningful description of players' behaviours as input—our method does not rely on any prior knowledge about player types and has the power to learn these tendencies based only on observation.

There are two potential issues with purely data-driven approaches. First, the statistical techniques used to construct data-driven models do not afford a semantically meaningful interpretation of outputs. For example, in survey-based models, players are often grouped into categories such as "explorer" or "achiever", both of which are human-interpretable. In contrast, the output of a statistical technique is a set of numbers indicating similarity to other players. While we may be able to cluster players together using these numbers, we cannot easily understand the behaviours that make the members of a cluster similar.

We do not view this as a limitation of our approach. Provided a predication that is useful in realizing a tailored game experience for players can be made, an inability to describe the qualities of players' behaviours is not a concern.

The second potential issue with a purely data-driven approach is that of algorithmic efficiency. The technique we describe below is designed to overcome this issue by breaking construction of models and making predictions into two, asynchronous, phases. This allows a model to be constructed offline where efficiency isn't as large a concern, and predictions to be made efficiently online where delays can affect players.

1.3 Aim and Objective- Problem Description

1.3.1. Aim

The aim is to find patterns from professional go game records (SGF format), their frequency and by removing repetition of its canonical forms by checking for symmetries.

1.3.2. Objective

To make this process simpler and more efficient, patterns are generated, not by hand but by writing simple scripts. Patterns are an efficient way of storing rules and can help the computer determine what to do when a board state similar to the pattern template is encountered.

Since the patterns are actually generated from previous game records of players, a certain assumption could be made on how another player would react, put in a similar situation as the recorded player.

Thus when the full computer Go game engine would be designed, these patterns generated would cut down a lot of computation, in the sense that a smaller game tree for next best move would now have to be searched.

1.4 Related Work:

The best performing Go programs through the ages have used the following techniques.

Goal Search

Instead of searching for max-min nodes, goal search looks for nodes in the game tree which accomplish specific tasks. In Go, these tasks include establishing life, death, eyes, connections, cuts, safety of territory, and captures. Since it is an early concept, it has not been proved effective..[11]

Heuristic Evaluation Functions

Despite the difficulty of creating heuristic evaluation functions, the best programs still use basic heuristics to direct search. For example, stones that are in “Atari” may lead the evaluator to value highly moves that save the stones. As a similar example, the evaluator may value moves which result in the capture of opponent’s stones and save the program’s own stones. Evaluation functions in the best programs are usually quite complex, involving estimations of life, death, and territory. Go researchers are also exploring algorithms for learning evaluation functions using machine learning techniques. For many of the best programs, these evaluation functions have proved useful as move ordering heuristics which cut down the effective branching factor of alpha-beta search.[12]

Monte-Carlo Tree Search

Monte-Carlo tree search has been one of the most effective strategies for playing computer Go. This evaluation plays out thousands of random games from potential move nodes in the game tree. The move chosen is the move which generates the best set of random games.

The intuition is that the heuristic favours moves which make connections and increase the liberties of groups, which is often a desired outcome for the player. Moreover, the Monte-Carlo evaluation function is cheaper to compute than most current board evaluation functions for Go, especially towards the middle and end of a typical game in which a winner can be determined in fewer random moves.

The problem with Monte-Carlo tree search is that it can easily overlook crucial moves, and in general tends to play very non-traditional moves. The game of Go involves a constant trade-off between exploration (establishing presence in unclaimed parts of the board) and exploitation (pursuing local battles for territory). To approach the exploration-exploitation problem, most current Monte-Carlo tree search programs use an algorithm called UCT (Upper bound Confidence for Tree), which was originally developed for the multi-armed bandit problem, to guide the tree search. [5]

Table Lookups

All of the most successful Go programs depend on storing tables of "*Joseki"* (sequences of moves that result in fair outcomes for both black and white), "*Fuseki"* (sequences of moves in the first few dozen moves of the game), and end-game manoeuvres in their programs. The obvious disadvantage of such a technique is that it cannot possibly store all the patterns and situations that may occur in a game.[12]

Neural Networks

Artificial neural networks have been used in several computer Go programs. In some cases, neural networks are used to learn evaluation functions, while in others, neural networks are employed for global and local pattern recognition. Due to the large board size and relatively few move restrictions in Go, many artificial neurons and connections are required to learn an evaluation function of any reasonable complexity. In addition, many layers are required to capture the complex interactions of stones in different parts of the board.[12]

Temporal Difference and Reinforcement Learning

Temporal difference (TD) learning and reinforcement learning proved to be successful game playing techniques for other games, including Backgammon. The large state space of Go prevents these techniques from being effective when applied directly to the game of Go. [12]

Cellular Automata

There has been some research in the applications of cellular automata to Go, due to the high resemblance of cellular automata to evolving Go board positions. In one paper, authors tried learning cellular automata rules from expert games. The number and type of different rules grew exponentially large, and so this type of learning was heavily space intensive, and moreover did not perform well in practice. While there has not been much success in this area, there is much ground yet to be explored.[12]

1.5 System Requirements

1.5.1 S/w requirements

- **Specific file format**- file should follow a specific format so that when scripts are run to extract relevant data the same code could be used universally. The .sgf files follow a particular order and are thus easy to read by the computer.
- **Simple board representation**- the software should be able to create a simple board state which resembles the actual go board so that moves could be analysed efficiently.
- **Choice of size and shape of pattern**- the same code is enough to give a 5X5 pattern as well as a 3X3 pattern with minimal changes. Shapes vary from square to diamond.
- **Pattern redundancy removal**- checking for the symmetry options using rotation, reflection

and flipping of colours to remove canonical forms.

- **Quick database retrieval system-** pattern matching is only possible fast if the pattern_db responds with the correct match of pattern quickly and efficiently.
- **Pattern matcher**- a pattern matching algorithm has to be devised which quickly evaluates which pattern should be used given that the current board state does match one of the previous recorded data pattern.

Stake holders

- **Amateur Go game players**: beginners want to learn the game using the software and the frequent and most widely used pattern search.
- **Game analysts**: many professional games are analysed and methods are devised to make the moves stronger to beat the opponent.
- **Game designers**: use the patterns to find a way to find the computers next move, because all the rules can be represented in the form of patterns.
- **Professional players:** new patterns and ways to beat a strong player requires knowledge of good opening moves so that a stable board state can be reached.
- **Go game teachers:** many teachers and instructors require patterns to make rules and subsequently teach the go student community.

1.5.2 H/w requirements

- **Memory constraints-** a basic memory constraint is there but we assume that since the professional game records are in a reasonable scale, memory for storage and backup is not a big problem. A normal desktop with normal configuration is preferable.
- **Processor constraints-** since a lot of computations are required to extract information, store them in a database, then retrieve it and create a normal board state, find patterns, find repetitions and then store unique patterns in the database, only to be retrieved when the system plays against a human player, thus a requirement of a good processor is a given.

1.6 Report Organization:

Section 2 shows the “Overview of the Proposed System” where the problem statement would be explained in detail with the related concepts. The shortcomings of the existing systems would be outlined and a solution would be presented.

Section 3 details the design and analysis part of the project report , wherein the functional and non-functional requirements would be enlisted. The design of the proposed system would also be detailed in this section and the method would be illustrated using “sequence model”, “use case model” and a “class diagram”.

Section 4 explains the Implementation part of my project. The tools used would be outlined and some modules which were recurring throughout my project would be shown with some code snippets. Testing cases would be designed and the pass/fail criteria would be found out.

Section 5 shows the results and the performance details.

2. Overview of the proposed System

2.1 Introduction of the problem and its related concepts

The inspiration of this paper comes from Nakamura’s work on pattern acquisition from game records using n-gram statistics [14]. An ***n*-gram** is a continuous sequence of *n* items from a given sequence of text or speech. An *n*-gram could be any combination of letters.

DRAWBACK: GO patterns are fundamentally different from words and word combinations in natural languages, because the game is played on a two-dimension game board. The organisation of words or combinations of words has a strict chronological order due to linearity of natural language as opposed to the Go patterns which are mostly played out of a particular sequence or order. Therefore only spatial relationships among the player’s moves on the board are of importance in the generation of patterns.

Although there are many patterns dictionaries available, the difference in their shapes and size limit the applicability.

Although large amount of GO patterns can be found in pattern dictionaries, they are not used in our experiments because dictionary patterns have different physical properties such as shape and size, making comparisons of their statistical usages meaningless. The moves in the records are shown as a board state and small segments are analysed and patterns are created. The fixed area used in our pattern definition is a 5-by-5 square, centred by the current move.

Additionally, the patterns are checked for their repetitiveness of the original patterns and their canonical forms. Shapes are considered to be equivalent if they are the same with respect to the rotation and flipping operations. Shapes are also considered to be equivalent with respect to colour switch. A maximum of 12 equivalent shapes are possible for a given pattern in its canonical form, among which four are obtained due to the rotation operation, two by reflection along x and y axis and the other six are due to the flipping operation.

The shapes are in the form of a 3x3, 5x5 and by a 5x5 in the form of a diamond. In the diamond they are sorted according to the information bits in it. Plus they are sorted according to their frequency of its usage in the other game records. Pattern templates are also created when one of the positions is subject to the “don’t care” condition and the don’t care position is checked whether ‘B’, ‘W’ and ‘_’ are all present in the ‘pattern_db’.

2.2 Gaps identified from the existing system

Though there are methods designed for patterns generation, there were no methods proposed to create patterns with “don’t care” conditions. So a simple algorithm was designed which made sure that the “don’t care” position was correctly checked and its repetitions are avoided by checking for its canonical forms too.

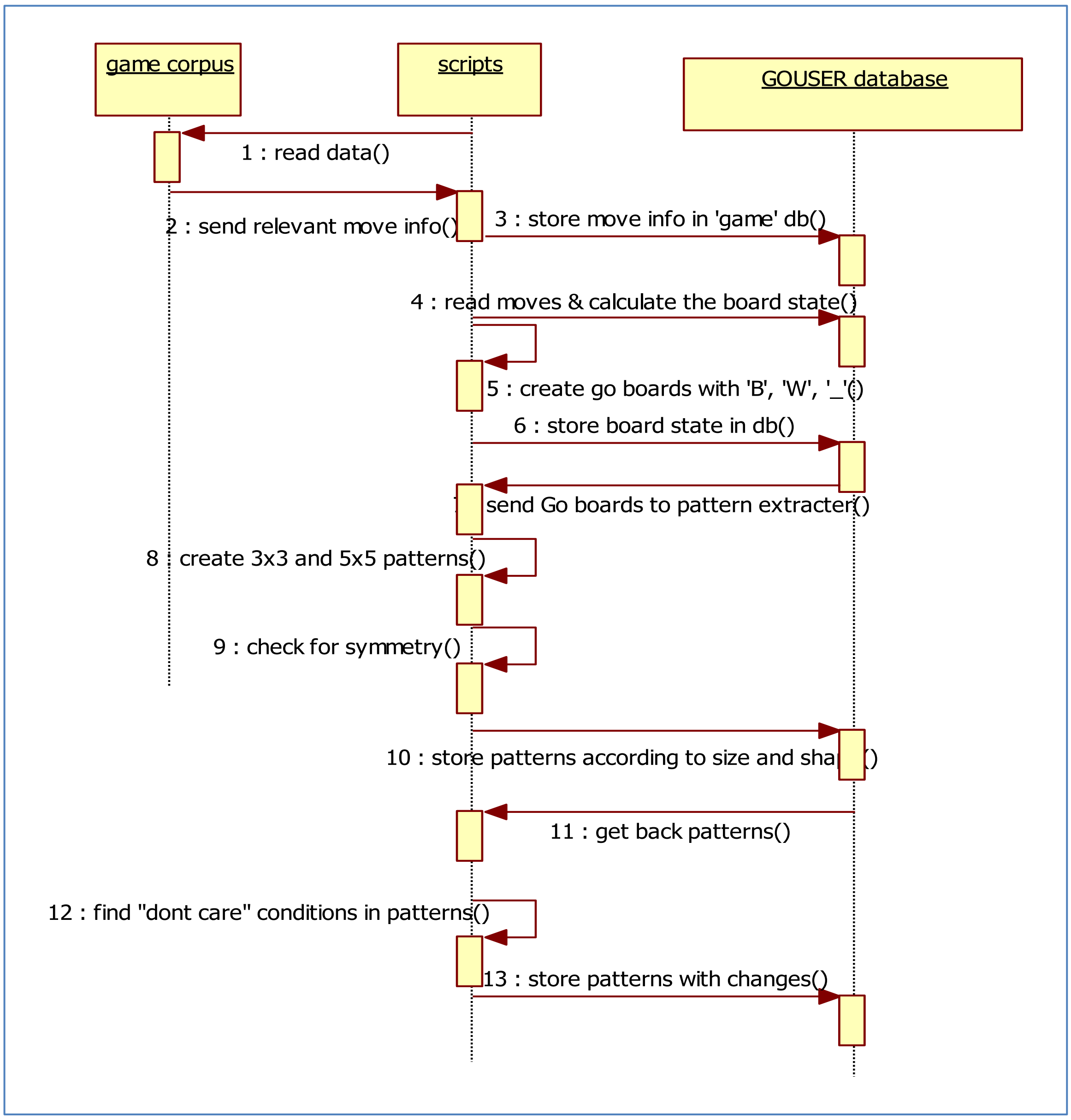
2.3 Proposed Solution

Pseudo-code:

```
   //finding the pattern template
char[] pattern_template=new char[9];  // a character array to store the pattern template.

for(int i=0;i<counter;i++)  // for each pattern(2d array)
{
        //for each position of the template from 0-8..
        //add a don’t care condition and then check the other
        // patterns and its variants for a match.
               // if for a particular don't care
               // ‘B’, 'W' and '_' all are present store the pattern template
               //else store the whole array.

    String st_pattern_template_B=new String();  // to compare strings
    String st_pattern_template_W=new String();  // to compare strings
    String st_pattern_template_blank=new String();  // to compare strings
    String st_pattern_template_temp=new String();

    for(int j=0;j<9;j++)
    {
           flag=0;
         pattern_template=st_pattern_array[i].toCharArray();
         pattern_template[j]='B';
         st_pattern_template_B=obj.oneDtoString(pattern_template);
         pattern_template[j]='W';
         st_pattern_template_W=obj.oneDtoString(pattern_template);
         pattern_template[j]='_';
         st_pattern_template_blank=obj.oneDtoString(pattern_template);

         pattern_template[j]='?';                                   //dont care template
         st_pattern_template_temp=obj.oneDtoString(pattern_template);

        for(int k=0;k<counter;k++)
        {
        x=((st_pattern_template_B.equals(st_pattern_array[k]))
         || (st_pattern_template_B.equals(st_pattern_array_inverse[k]))
         || (st_pattern_template_B.equals(st_pattern_array_90[k]))
         || (st_pattern_template_B.equals(st_pattern_array_inverse_90[k]))
         || (st_pattern_template_B.equals(st_pattern_array_180[k]))
         || (st_pattern_template_B.equals(st_pattern_array_inverse_180[k]))
         || (st_pattern_template_B.equals(st_pattern_array_270[k]))
         || (st_pattern_template_B.equals(st_pattern_array_inverse_270[k]))
         || (st_pattern_template_B.equals(st_pattern_array_x[k]))
         || (st_pattern_template_B.equals(st_pattern_array_inverse_x[k]))
         || (st_pattern_template_B.equals(st_pattern_array_y[k]))
         || (st_pattern_template_B.equals(st_pattern_array_inverse_y[k])));
if(x)
{
    for(int k1=0;k1<counter;k1++)
    {
        y=((st_pattern_template_W.equals(st_pattern_array[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_inverse[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_90[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_inverse_90[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_180[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_inverse_180[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_270[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_inverse_270[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_x[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_inverse_x[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_y[k1]))
         || (st_pattern_template_W.equals(st_pattern_array_inverse_y[k1]))
        );
if(y)
{
    for(int k2=0;k2<counter;k2++)
    {
        z=((st_pattern_template_blank.equals(st_pattern_array[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_inverse[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_90[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_inverse_90[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_180[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_inverse_180[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_270[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_inverse_270[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_x[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_inverse_x[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_y[k2]))
         || (st_pattern_template_blank.equals(st_pattern_array_inverse_y[k2])));

                 if(z)
                 {
                     flag=1;
                     break;
                 }}}}}}

        if(flag==1)
        {
            pattern_3by3_final[d]=st_pattern_template_temp;
            flag=0;
            d++;        }        } }
System.out.println("the pattern template found.........with repetitions"+d);
for(int k=0;k<d;k++)
{
    System.out.println("....count.....  "+k);
    System.out.println();
    obj1.print2D_from1D(pattern_3by3_final[k]);
    System.out.println();
}
    for(int i=0;i<d;i++)
    {
       pattern_array=obj2.stringto2D(pattern_3by3_final[i]);
       flag=0;

       pattern_array_inverse=obj1.inverse(pattern_array);
       pattern_array_90=obj1.rotate(pattern_array);
       pattern_array_inverse_90=obj1.inverse(pattern_array_90);
       pattern_array_180=obj1.rotate(pattern_array_90);
       pattern_array_inverse_180=obj1.inverse(pattern_array_180);
       pattern_array_270=obj1.rotate(pattern_array_180);
       pattern_array_inverse_270=obj1.inverse(pattern_array_270);
       pattern_array_x=obj1.reflection_x(pattern_array); //reflection along x axis
       pattern_array_y=obj1.reflection_y(pattern_array); //reflection along y axis
       pattern_array_inverse_x=obj1.inverse(pattern_array_x);
       pattern_array_inverse_y=obj1.inverse(pattern_array_y);

       st1_pattern_array=obj1.charToString(pattern_array);
       st1_pattern_array_inverse=obj1.charToString(pattern_array_inverse);
       st1_pattern_array_90=obj1.charToString(pattern_array_90);
       st1_pattern_array_inverse_90=obj1.charToString(pattern_array_inverse_90);
       st1_pattern_array_180=obj1.charToString(pattern_array_180);
       st1_pattern_array_inverse_180=obj1.charToString(pattern_array_inverse_180);
       st1_pattern_array_270=obj1.charToString(pattern_array_270);
       st1_pattern_array_inverse_270=obj1.charToString(pattern_array_inverse_270);
       st1_pattern_array_x=obj1.charToString(pattern_array_x);
       st1_pattern_array_inverse_x=obj1.charToString(pattern_array_inverse_x);
       st1_pattern_array_y=obj1.charToString(pattern_array_y);
       st1_pattern_array_inverse_y=obj1.charToString(pattern_array_inverse_y);

       for(int l=0;l<d1;l++)
          {
           if((st_pattern_array_db[l].equals(st1_pattern_array))||
              (st_pattern_array_db[l].equals(st1_pattern_array_inverse))||
              (st_pattern_array_db[l].equals(st1_pattern_array_90))||
              (st_pattern_array_db[l].equals(st1_pattern_array_inverse_90))||
              (st_pattern_array_db[l].equals(st1_pattern_array_180))||
              (st_pattern_array_db[l].equals(st1_pattern_array_inverse_180))||
              (st_pattern_array_db[l].equals(st1_pattern_array_270))||
              (st_pattern_array_db[l].equals(st1_pattern_array_inverse_270))||
        (st_pattern_array_db[l].equals(st1_pattern_array_x))||
        (st_pattern_array_db[l].equals(st1_pattern_array_inverse_x))||
        (st_pattern_array_db[l].equals(st1_pattern_array_y))||
        (st_pattern_array_db[l].equals(st1_pattern_array_inverse_y)))
        {
          flag=1;break;
        }
      }

  if(flag==0)
  {
     st_pattern_array_db[d1]=obj1.charToString(pattern_array);
     d1++;
  }
}
```

3. Analysis and Design

3.1. Brief Introduction



game corpus
scripts
GOUSER database
1 : read data()
2 : send relevant move info()
3 : store move info in 'game' db()
4 : read moves & calculate the board state()
5 : create go boards with 'B', 'W', '_'()
6 : store board state in db()
send Go boards to pattern extracter()
8 : create 3x3 and 5x5 patterns()
9 : check for symmetry()
10 : store patterns according to size and sha()
11 : get back patterns()
12 : find "dont care" conditions in patterns()
13 : store patterns with changes()

Fig1. Sequence diagram to show the whole pattern generation algorithm

3.2. Requirement analysis

3.2.1. Functional requirements

3.2.1.1. **Specific file format**- file should follow a specific format so that when scripts are run to extract relevant data the same code could be used universally. The .sgf files follow a particular order and are thus easy to read by the computer.

3.2.1.2. **Simple board representation**- the software should be able to create a simple board state which resembles the actual go board so that moves could be analysed efficiently

3.2.1.3. **Choice of size and shape of pattern**- the same code is enough to give a 5X5 pattern as well as a 3X3 pattern with minimal changes. Shapes vary from square to diamond

3.2.1.4. **Pattern redundancy removal**- checking for the symmetry options using rotation, reflection and flipping of colours to remove canonical forms

3.2.1.5. **Quick database retrieval system**- pattern matching is only possible fast if the 'pattern_db' responds with the correct match of pattern quickly and efficiently.

3.2.1.6. **Pattern matcher**- a pattern matching algorithm has to be devised which quickly evaluates which pattern should be used given that the current board state does match one of the previous recorded data pattern.

3.2.2. Non-functional requirements

3.2.2.1. **Usability** Since all users are familiar with the general usage of automobile management, no specific training is required.The system is user friendly and self-explanatory.

3.2.2.2. **Reliability**

3.2.2.2.1. **Mean Time between Failures (MTBF)** : the system will be developed in such a manner that it will fail once a year.

3.2.2.2.2. **Mean Time to Repair (MTTR):**Even if the system fails, the system will be recovered back up within an hour or less.

3.2.2.2.3. **Accuracy:** The accuracy of the system is limited by the accuracy of the speed at which the employees use the system.

3.2.2.3. **Performance**

3.2.2.3.1. **Response Time**: Various pages of the system should process in seconds and the interaction with the server should be as less as possible to decrease response time.

3.2.2.3.2. **Resource Utilization:**The resources are modified according the user requirements and also according to the facilities requested by the users.

3.3. Design of the proposed system

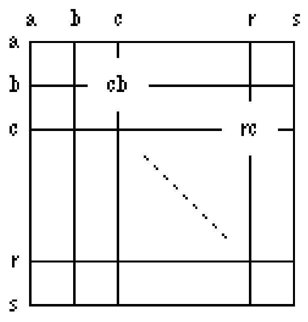
3.3.1. Design

3.3.1.1. Design overview (methodology)

3.3.1.1.1. Data Integration: First of all the data are collected and integrated from all the different sources. The data was collected from sources such as http://www.go4go.net/go, http://www.u-go.net/links/gamerecords/ etc. These files are in the SGF file format [1]. Some files are even sorted player and tournament wise.

SGF FILE SPECIFICATION:

In Go the Stone becomes Point and the Move and Point type are the same: two lowercase letters.



Coordinate system for points and moves

The first letter designates the column (left to right), the second the row (top to bottom). The upper left part of the board is used for smaller boards, e.g. letters "a"-"m" for 13*13. A pass move is shown as '[]' or alternatively as '[tt]' (only for boards <= 19x19), i.e. applications should be able to deal with both representations.

3.3.1.2. **Data Selection:** in this step select only those data which is useful for data mining. Of all the various data fields that we obtain, only the required data fields is to be selected.

OUTPUT:

```
**mysql> desc game;
+-----------+-----------+------+-----+---------+-------+
| Field     | Type      | Null | Key | Default | Extra |
+-----------+-----------+------+-----+---------+-------+
| file_name | char(100) | NO   | PRI |         |       |
| file_info | text      | YES  |     | NULL    |       |
+-----------+-----------+------+-----+---------+-------+
**mysql> desc move;
+--------------+-----------+------+-----+---------+-------+
| Field        | Type      | Null | Key | Default | Extra |
+--------------+-----------+------+-----+---------+-------+
| file_name    | char(20)  | NO   | PRI | NULL    |       |
| player_white | char(20)  | YES  |     | NULL    |       |
| player_black | char(20)  | YES  |     | NULL    |       |
| komi         | char(10)  | YES  |     | NULL    |       |
| result       | char(100) | YES  |     | NULL    |       |
| move_made    | text      | YES  |     | NULL    |       |
+--------------+-----------+------+-----+---------+-------+
**mysql> desc board_state;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| file_name | char(20) | NO   | PRI |         |       |
| state     | blob     | YES  |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
```

3.3.1.1.3. Data Cleaning: The data we have collected are not clean and may contain errors, missing values, noisy or inconsistent data. So we need to apply different techniques to get rid of such anomalies.

```
OUTPUT:
mysql> select * from move where file_name="1.sgf";
+-----------+--------------+--------------+---------+----------+--------------------------------------------------
| file_name | player_white | player_black | komi    | result   | move_made|
+-----------+--------------+--------------+---------+----------+--------------------------------------------------
| 1.sgf     | WR[8p]       | BR[8p]       | KM[5.5] | RE[B+2.5] |
;B[qd];W[dc];B[dp];W[pq];B[ce];W[oc];B[ld];W[of];B[oe];W[ne];B[pe];W[md];B[nf];W[le];B[mf];W[k
d];B[og];W[qn];B[lc];W[me];B[kc];W[jd];B[jc];W[id];B[mb];W[nb];B[mc];W[nc];B[hb]
;W[gc];B[hc];W[hd];B[gd];W[de];B[df];W[ee];B[cd];W[gb];B[ic];W[ge];B[jq];W[cc];B
[ci];W[cq];B[cp];W[dq];B[eq];W[er];B[fq];W[fr];B[gq];W[bp];B[bo];W[bq];B[cn];W[e
i];B[fd];W[fe];B[ql];W[on];B[pj];W[mq];B[qq];W[pr];B[mp];W[lp];B[nq];W[lq];B[pp]
;W[op];B[po];W[pn];B[ro];W[rn];B[qr];W[oo];B[qo];W[nr];B[rm];W[sn];B[rq];W[ek];B
```

```
[ck];W[gr];B[hp];W[rk];B[qk];W[rj];B[qi];W[rl];B[lf];W[kf];B[kg];W[jf];B[ri];W[q
b];B[rc];W[hr];B[jo];W[ch];B[dh];W[di];B[bh];W[cj];B[bi];W[eh];B[cg];W[ji];B[gl]
;W[em];B[gj];W[eo];B[ep];W[li];B[fn];W[lg];B[ih];W[gh];B[ii];W[ng];B[pf];W[mg];B
[of];W[jg];B[jj];W[kj];B[jk];W[iq];B[oq];W[np];B[kn];W[ip];B[kq];W[io];B[in];W[r
b];B[ps];W[or];B[qm];W[sm];B[nh];W[ni];B[mh];W[lh];B[oh];W[kl];B[jm];W[ho];B[dj]
;W[dk];B[bj];W[cl];B[bl];W[hm];B[hn];W[gm];B[gn];W[fm];B[kk];W[lk];B[jl];W[cm];B
[bm];W[ln];B[lo];W[lm];B[ga];W[fc];B[fb];W[ec];B[gi];W[fk];B[ko];W[mo];B[ml];W[l
l];B[nk];W[nj];B[bc];W[bb];B[bd];W[gk];B[jh];W[do];B[ki];W[lj];B[eg];W[hj];B[hi]
;W[hl];B[dm];W[dl];B[fh];W[fi];B[fg];W[dn];B[hf];W[ab];B[en];W[co];B[bn];W[fo];B
[go];W[fl];B[sb];W[qc];B[re];W[ma];B[la];W[na];B[ja];W[lr];B[jr];W[ol];B[ok];W[n
l];B[mk];W[mm];B[od];W[nd];B[eb];W[db];B[fa];W[gf];B[gg];W[hg];B[ig];W[if];B[kh]
;W[os];B[qs];W[so];B[sp];W[ac];B[ad];W[he];B[ij];W[si];B[sh];W[sj];B[sl];W[sk];B
[dd];W[ed];B[oi];W[hk];B[pc];W[pb];B[ap];W[aq];B[ao];W[hf];B[hh];W[pm];B[pd];W[l
s];B[is];W[hs];B[pl];W[oj];B[ra];W[sl];B[nm];W[om];B[qj];W[mj];B[pk];W[rh];B[sg]
;W[sd];B[sc];W[rf];B[sf];W[se];B[rd];W[se];B[sd];W[qh];B[pi];W[qf];B[qe];W[qg];B
[pg];W[qa];B[km];W[da];B[sa];W[ej];B[cj];W[hq];B[gp];W[ef];B[im]) |
>>1 row in set (0.04 sec)
```

3.3.1.1.4. Data Transformation: The data even after cleaning are not ready for mining as we need to transform them into forms appropriate for mining. The techniques used to accomplish this are smoothing, aggregation, normalization etc.

```
OUTPUT:

run:
connection made
statement executed
result object got

FILE====>1.sgf
MOVE====>;B[qd];W[dc];B[dp];W[pq];B[ce];W[oc];B[ld];W[of];B[oe];W[ne];B[pe];W[md];B[
nf];W[le];B[mf];W[kd];B[og];W[qn];B[lc];W[me];B[kc];W[jd];B[jc];W[id];B[mb];W[nb];B[mc];W[nc
];B[hb];W[gc];B[hc];W[hd];B[gd];W[de];B[df];W[ee];B[cd];W[gb];B[ic];W[ge];B[jq];W[cc];B[ci];W
[cq];B[cp];W[dq];B[eq];W[er];B[fq];W[fr];B[gq];W[bp];B[bo];W[bq];B[cn];W[ei];B[fd];W[fe];B[ql];
W[on];B[pj];W[mq];B[qq];W[pr];B[mp];W[lp];B[nq];W[lq];B[pp];W[op];B[po];W[pn];B[ro];W[rn];B
[qr];W[oo];B[qo];W[nr];B[rm];W[sn];B[rq];W[ek];B[ck];W[gr];B[hp];W[rk];B[qk];W[rj];B[qi];W[rl]
;B[lf];W[kf];B[kg];W[jf];B[ri];W[qb];B[rc];W[hr];B[jo];W[ch];B[dh];W[di];B[bh];W[cj];B[bi];W[eh]
;B[cg];W[ji];B[gl];W[em];B[gj];W[eo];B[ep];W[li];B[fn];W[lg];B[ih];W[gh];B[ii];W[ng];B[pf];W[mg
];B[of];W[jg];B[jj];W[kj];B[jk];W[iq];B[oq];W[np];B[kn];W[ip];B[kq];W[io];B[in];W[rb];B[ps];W[or
];B[qm];W[sm];B[nh];W[ni];B[mh];W[lh];B[oh];W[kl];B[jm];W[ho];B[dj];W[dk];B[bj];W[cl];B[bl];
W[hm];B[hn];W[gm];B[gn];W[fm];B[kk];W[lk];B[jl];W[cm];B[bm];W[ln];B[lo];W[lm];B[ga];W[fc];B
[fb];W[ec];B[gi];W[fk];B[ko];W[mo];B[ml];W[ll];B[nk];W[nj];B[bc];W[bb];B[bd];W[gk];B[jh];W[do]
;B[ki];W[lj];B[eg];W[hj];B[hi];W[hl];B[dm];W[dl];B[fh];W[fi];B[fg];W[dn];B[hf];W[ab];B[en];W[co]
;B[bn];W[fo];B[go];W[fl];B[sb];W[qc];B[re];W[ma];B[la];W[na];B[ja];W[lr];B[jr];W[ol];B[ok];W[nl
];B[mk];W[mm];B[od];W[nd];B[eb];W[db];B[fa];W[gf];B[gg];W[hg];B[ig];W[if];B[kh];W[os];B[qs];
W[so];B[sp];W[ac];B[ad];W[he];B[ij];W[si];B[sh];W[sj];B[sl];W[sk];B[dd];W[ed];B[oi];W[hk];B[pc]
;W[pb];B[ap];W[aq];B[ao];W[hf];B[hh];W[pm];B[pd];W[ls];B[is];W[hs];B[pl];W[oj];B[ra];W[sl];B[
nm];W[om];B[qj];W[mj];B[pk];W[rh];B[sg];W[sd];B[sc];W[rf];B[sf];W[se];B[rd];W[se];B[sd];W[qh]
;B[pi];W[qf];B[qe];W[qg];B[pg];W[qa];B[km];W[da];B[sa];W[ej];B[cj];W[hq];B[gp];W[ef];B[im])

    length===1759
```

```
    printing the int array
    0 2 2 1 0 0 0 0 0 0 0 0 0 0 1 1 2 0 0
    0 2 1 1 0 0 0 1 1 1 0 1 1 1 1 2 2 0 0
    0 0 2 1 1 0 1 2 1 1 1 2 2 1 2 1 2 0 0
    2 2 2 1 2 1 0 1 2 1 2 2 1 2 2 1 2 0 0
    0 1 2 2 2 2 1 2 2 2 2 0 2 1 2 1 1 2 0
    1 1 2 1 2 0 1 1 2 0 2 2 2 1 2 0 1 2 0
    1 2 2 1 2 2 1 2 1 1 2 1 2 1 1 1 1 2 0
    0 1 1 2 2 2 2 1 1 2 2 2 2 1 2 1 2 2 2
    0 0 1 2 0 2 1 1 1 1 0 0 1 1 2 2 2 0 1
    1 0 1 2 0 2 2 1 2 1 1 1 1 0 1 0 1 1 0
    0 0 1 2 0 2 1 1 1 2 1 2 1 1 1 0 1 0 0
    1 0 1 1 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2
    2 1 1 2 2 1 2 1 0 2 1 1 2 0 2 1 2 0 0
    2 2 2 2 2 1 2 1 2 2 1 2 1 0 0 2 1 2 0
    0 0 2 1 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2
    0 2 1 1 1 1 1 0 1 1 1 1 2 2 1 1 2 2 1
    2 2 2 1 1 2 2 2 1 1 1 1 1 2 1 0 1 1 1
    1 2 1 1 1 2 0 2 1 2 2 2 1 2 1 0 1 0 0
    1 1 1 1 2 1 1 1 2 2 2 2 2 2 2 1 0 0 0
```

```
    FILE====>10.sgf
    MOVE====>;B[pd];W[dc];B[dq];W[qp];B[de];W[ce];B[cf];W[cd];B[op];W[df];B[pn];W[do];B[e
o];W[en];B[ep];W[dn];B[cp];W[oq];B[nq];W[pq];B[nr];W[rn];B[fn];W[fm];B[gm];W[gn];B[fo];W[fl]
;B[gl];W[fk];B[hn];W[np];B[pp];W[qq];B[mp];W[no];B[on];W[nn];B[ol];W[nm];B[pl];W[nl];B[nk];
W[mk];B[nj];W[lp];B[lq];W[mo];B[kp];W[mq];B[mr];W[mj];B[ni];W[mi];B[nh];W[gk];B[mp];W[rl];
B[lo];W[mh];B[ll];W[ml];B[ng];W[ef];B[jc];W[bo];B[gc];W[lf];B[di];W[cj];B[ci];W[bj];B[dj];W[fi];
B[cl];W[ck];B[bm];W[dk];B[co];W[qf];B[qe];W[pf];B[mf];W[od];B[oe];W[pe];B[qd];W[oc];B[pb];W[
pc];B[qc];W[ob];B[qb];W[re];B[rd];W[kc];B[kd];W[ld];B[lc];W[kb];B[le];W[md];B[me];W[lb];B[rj];
W[ke];B[jd];W[kf];B[jb];W[mc];B[mg];W[lg];B[qj];W[ne];B[of];W[hd];B[hc];W[fb];B[hf];W[ge];B[fc
];W[eb];B[fe];W[gf];B[ff];W[gg];B[fg];W[fh];B[ee];W[cg];B[id];W[gd];B[jf];W[kh];B[jh];W[hh];B[ji
];W[jj];B[bf];W[dg];B[ij];W[jk];B[ki];W[ik];B[fd];W[je];B[dd];W[cb];B[ec];W[hb];B[gb];W[ga];B[ie
];W[ig];B[ib];W[ha];B[fa];W[ea];B[ia];W[fa];B[bg];W[bh];B[ch];W[eh];B[bc];W[cc];B[bb];W[ba];B
[da];W[db];B[be];W[bd];B[ad];W[ca];B[ac];W[ae];B[af])
```

```
length===1063
```

```
printing the int array
0 0 1 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 1 2 1 1 1 2 0 2 0 0 1 0 2 0 0 0 0
2 2 2 2 2 1 2 1 1 2 2 1 0 0 1 1 0 0 0
1 2 2 1 1 2 2 0 1 1 2 0 0 2 2 0 1 0 0
2 2 1 0 1 2 0 2 0 0 0 0 0 2 1 1 0 0 0
2 2 1 1 1 1 1 2 2 0 2 2 2 1 1 0 0 0 0
2 1 1 2 2 2 2 0 0 0 2 1 1 2 0 0 0 0 0
2 2 1 2 0 1 0 2 0 0 0 0 0 1 0 0 0 0 0
1 1 0 1 1 0 2 0 0 1 2 0 0 0 0 0 0 0 0
0 1 1 1 2 1 0 1 1 2 2 0 0 0 0 0 0 0 0
0 2 2 1 2 2 0 2 1 0 0 0 0 0 0 1 0 0 0
0 2 1 2 1 2 2 0 0 0 0 1 0 0 1 2 1 0 0
0 0 2 2 1 1 1 2 2 2 2 2 0 0 2 1 2 1 0
0 0 0 0 2 0 1 1 1 1 1 2 2 2 2 2 1 1 0
0 2 2 2 1 1 0 0 0 0 0 1 0 1 0 1 2 0 0
0 1 2 1 2 2 0 0 0 0 0 1 0 1 0 1 2 0 0
0 1 1 1 1 2 0 0 0 1 0 0 0 0 0 2 2 0 0
0 0 0 1 2 0 0 0 0 1 0 2 0 2 0 0 0 0 0
0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

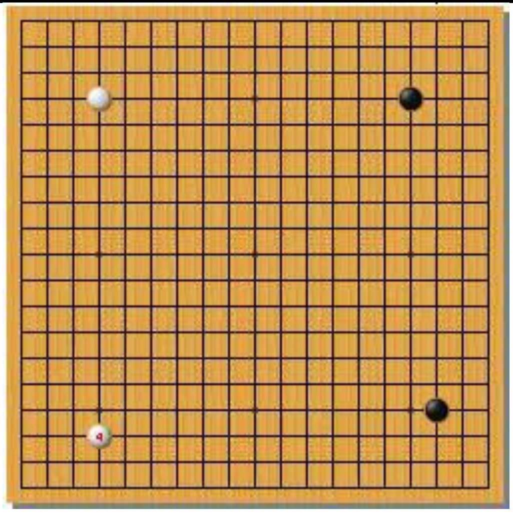
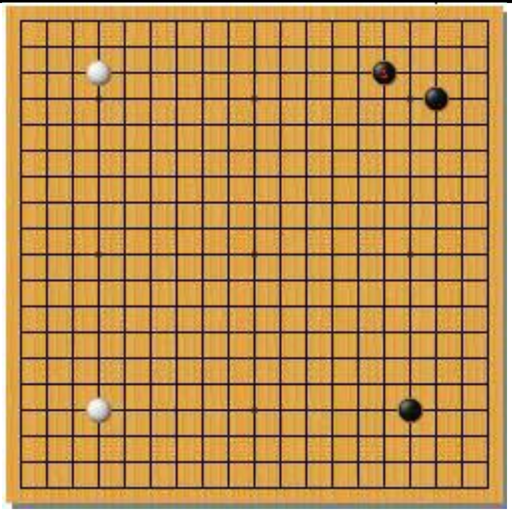
3.3.1.1.5. Data Mining: Now we are ready to apply data mining techniques on the data to discover the interesting patterns. Techniques like clustering and association analysis are among the many different techniques used for data mining.

```
OUTPUT:
```

```
mysql> select file_name , move_made from move where move_made like"_B[qd]_W[ce]_B[pq]_W[od]%";
| file_name | move_made |
+-------------+------------------------------------------------------------------
| 6388.sgf   |

 ;B[qd];W[ce];B[pq];W[od];B[oc];W[nc];B[pc];W[md];B[qf];W[gc];B[cp]
;W[eq];B[hp];W[go];B[dn];W[cq];B[bq];W[dp];B[co];W[ho];B[jp];W[ip];B[iq];W[io];B
[hq];W[lp];B[jo];W[jn];B[ko];W[po];B[np];W[no];B[op];W[mo];B[kn];W[km];B[jm];W[i
n];B[lm];W[kl];B[ll];W[oo];B[qp];W[ln];B[kq];W[lq];B[fp];W[lk];B[nl];W[kk];B[pl]
;W[ol];B[om];W[nm];B[ok];W[ml];B[fq];W[br];B[dr];W[er];B[cr];W[dq];B[bs];W[ar];B
[aq];W[eo];B[lo];W[pp];B[mn];W[qq];B[qr];W[rq];B[or];W[rr];B[mr];W[qs];B[nn];W[e
n];B[dl];W[dj];B[jd];W[ge];B[lc];W[ld];B[kd];W[lb];B[kb];W[mc];B[kg];W[kc];B[jc]
;W[ng];B[ig];W[ie];B[je];W[if];B[mh];W[pe];B[pf];W[jf];B[kf];W[le];B[ke];W[qe];B
```

```
[re];W[of];B[ne];W[oe];B[rg];W[mg];B[lh];W[jg];B[jh];W[ih];B[ji];W[hh];B[il];W[i
k];B[gl];W[el];B[ek];W[fl];B[hk];W[ij];B[fk];W[dk];B[dm];W[hm];B[fn];W[fo];B[em]
;W[hl];B[cc];W[ec];B[cf];W[cd];B[ei];W[di];B[eh];W[dh];B[dc];W[df];B[eb];W[fb];B
[ed];W[fc];B[bd];W[be];B[bb];W[mp];B[lr];W[qm];B[nb];W[mb];B[ob];W[jb];B[bk];W[k
a];B[ql];W[rl];B[rk];W[ad];B[ac];W[bc];B[rn];W[rm];B[bd];W[rj];B[qk];W[bc];B[qn]
;W[pn];B[bd];W[ni];B[mj];W[bc];B[pm];W[ro];B[bd];W[mi];B[lj];W[bc];B[sl];W[qo];B
[bd];W[kj];B[li];W[bc];B[cg];W[dg];B[bd];W[mk];B[nh];W[bc];B[og];W[ab];B[sm];W[b
j];B[ck];W[ej];B[fj];W[fi];B[gi];W[fh];B[gh];W[fg];B[hj];W[bl];B[cl];W[ak];B[gg]
;W[gf];B[cj];W[bi];B[ci];W[ch];B[hg];W[bm];B[gn];W[jl];B[bn];W[pr];B[ic];W[nq];B
[nr];W[oq];B[jj];W[jk];B[ii];W[hs];B[is];W[hb];B[al];W[am];B[mm];W[bh];B[hd];W[g
d];B[an];W[aj];B[pd];W[lf];B[gs];W[gr];B[hr];W[fr];B[cm];W[al];B[ib];W[ia];B[so]
;W[gm];B[fm];W[ds];B[as];W[sp];B[lg];W[nf];B[ki];W[im];B[ma];W[nk];B[nj];W[os];B
[ns];W[kp];B[sn];W[ps]) |
```

3.3.1.1.6. Pattern Evaluation and Knowledge Presentation: This step involves visualization, transformation, removing redundant patterns etc. from the patterns we generated.

OUTPUT: FUSEKI PATTERNS	
	

3.3.1.1.7. Decisions / Use of Discovered Knowledge: This step helps user to make use of the knowledge acquired to take better decisions.

Fuseki pattern
1. Characteristics
• **Less systematic :** *Since each move is typically isolated and unforced, patterns for play on the whole board have seen much less systematic study than for Joseki, which are often contact moves which require specific and immediate responses. Hence a game of Go may easily explore an unfamiliar path.*   • **Recognised names :** *Only a proportion of fusekis have recognised or specific names. These include the two-star fuseki (nirensei fuseki), three-star fuseki (sanrensei fuseki), Chinese fuseki, Kobayashi fuseki,*

andShusaku fuseki. These are names for the influential formations which Black makes on one side of the board.

2. Type of fuseki

- **Territorial approach:** *As played on a large board (i.e. the standard 19x19 line goban), traditional wisdom says the priority is to play corner enclosures, then to extend to the middle of the sides, and finally to the center because it is easier to secure territory in the corners than on the sides or in the center. The classical view, particularly for the 3-3, 3-4 or 4-3 point, emphasizes good points to play in the opening because these points ensure larger and/or faster corner enclosure. Higher points are discouraged. This approach has clearer goals (control territory in the corners) and is easier for beginners to grasp and play.*

- **Influence-oriented approach:** *Unlike the territory-oriented playing style, this approach emphasizes control of the center. The reason for this is that one's play should not be narrowly focused on attempting to secure points quickly by occupying the corners first. Although it requires more effort to secure the center, it constitutes the majority of territory on the board. The key is to build a good framework in order to control the center of the board. Higher points like 4-4, 4-5 or 5-4 are encouraged. Some players occupy the side very quickly in order to build up a good framework, while some place their stones around the center. However, the influence-oriented approach is more abstract and harder for beginners to grasp and play.*

JOSEKI patterns

Joseki are "sequences" of moves which have been

- played and documented in high-level play, and
- studied and deemed as consisting of optimal (balanced) moves for both sides.
- They are studied sequences of moves in the corner areas of the Go board, for which the result is considered *balanced* for both black and white sides. Because games typically start with plays in the corners, players often try to use their understanding of joseki to gain local advantages in the corners — advantages which can in turn make for a better overall position.

Architecture

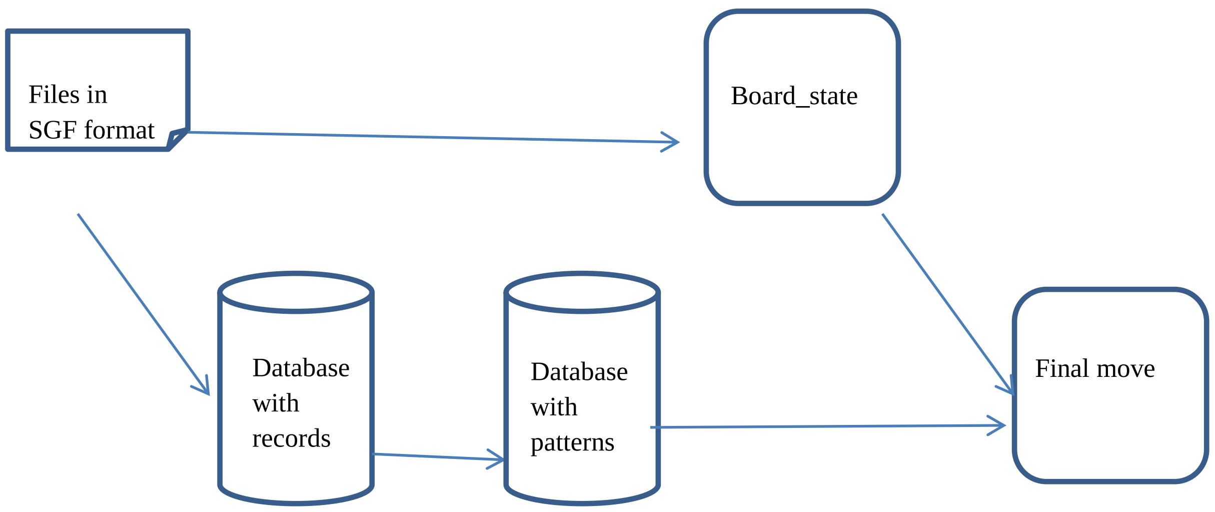


Fig 2: broad view of the data movement

3.3.2. Use case diagram

3.3.2.1. Actors

- **Professional game records**- these are digitized game corpuses which contain 36000+ professional games in the SGF format.
- **Database 'GOUSER'**- this is the database collection which has tables 'board_state', 'fuseki', 'pattern_db', 'move', 'game'.
- **Programmer**- does the data extraction , scripting for the pattern generation and stores them into the database
- **Human player**- uses his game knowledge to make the move after the system has made his move on the go game board.
- **Computer System**- system knowledge database is in the form of a pattern database which contains the different types and shapes of the patterns (both square and diamond) in sizes of 3x3 and 5x5. It uses the patterns to match the board state currently under play and based on the pattern makes the best possible move.

3.3.2.2. Use cases

- store player and game information
- get player and game information
- extract move information → create unordered 19x19 board state
- create move patterns → get size and shape
- check for its canonical forms → check for symmetry by rotation, reflection and by colour flipping
- remove pattern repetitiveness
- store pattern in database table 'pattern_db'
- get board state
- check for pattern similarity on the current board with the pattern database
- make a valid move based on pattern knowledge
- human player makes a move based on his go knowledge.

3.3.2.3. Diagram

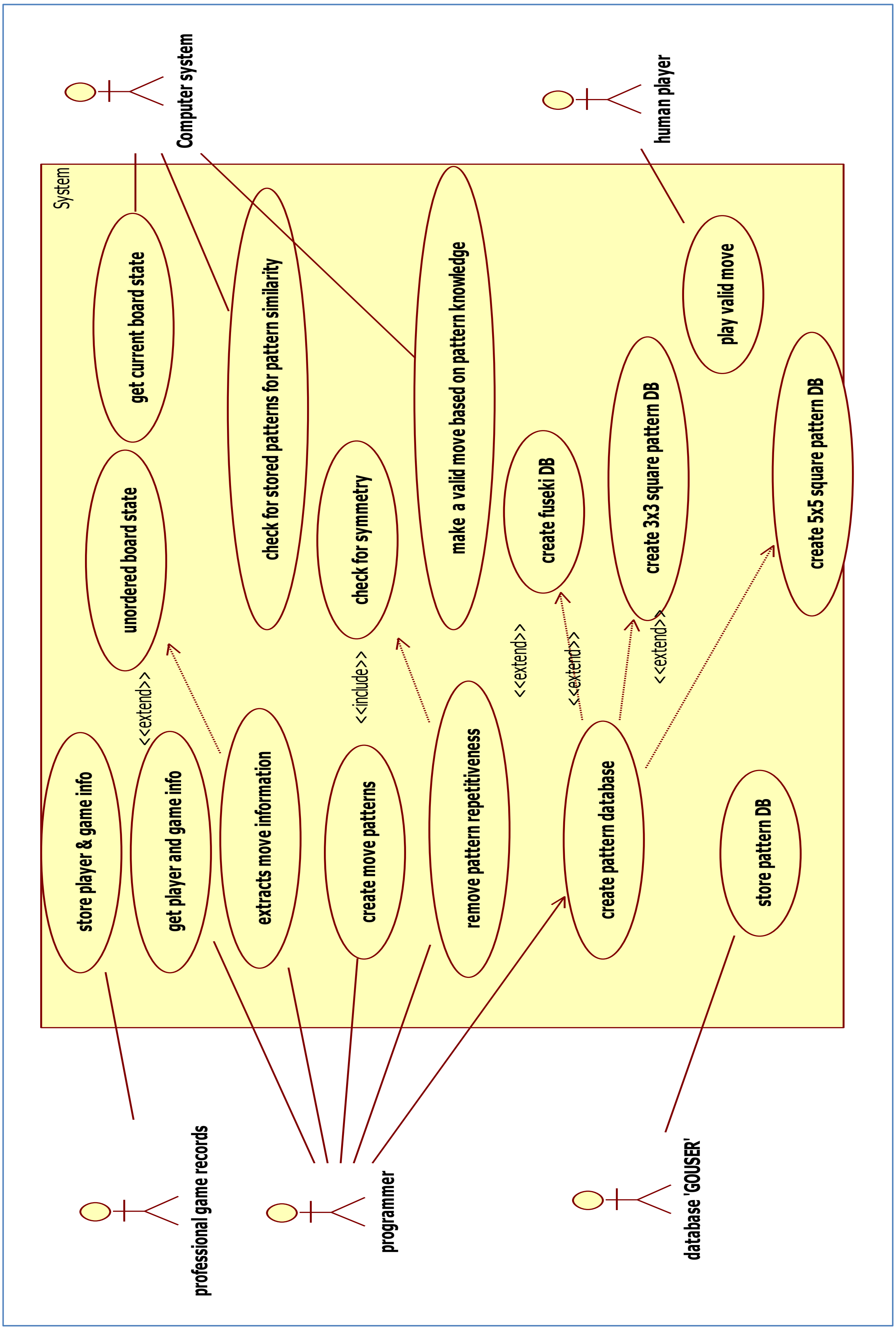


Fig 3. Use case model

3.3.3. Class-diagram:

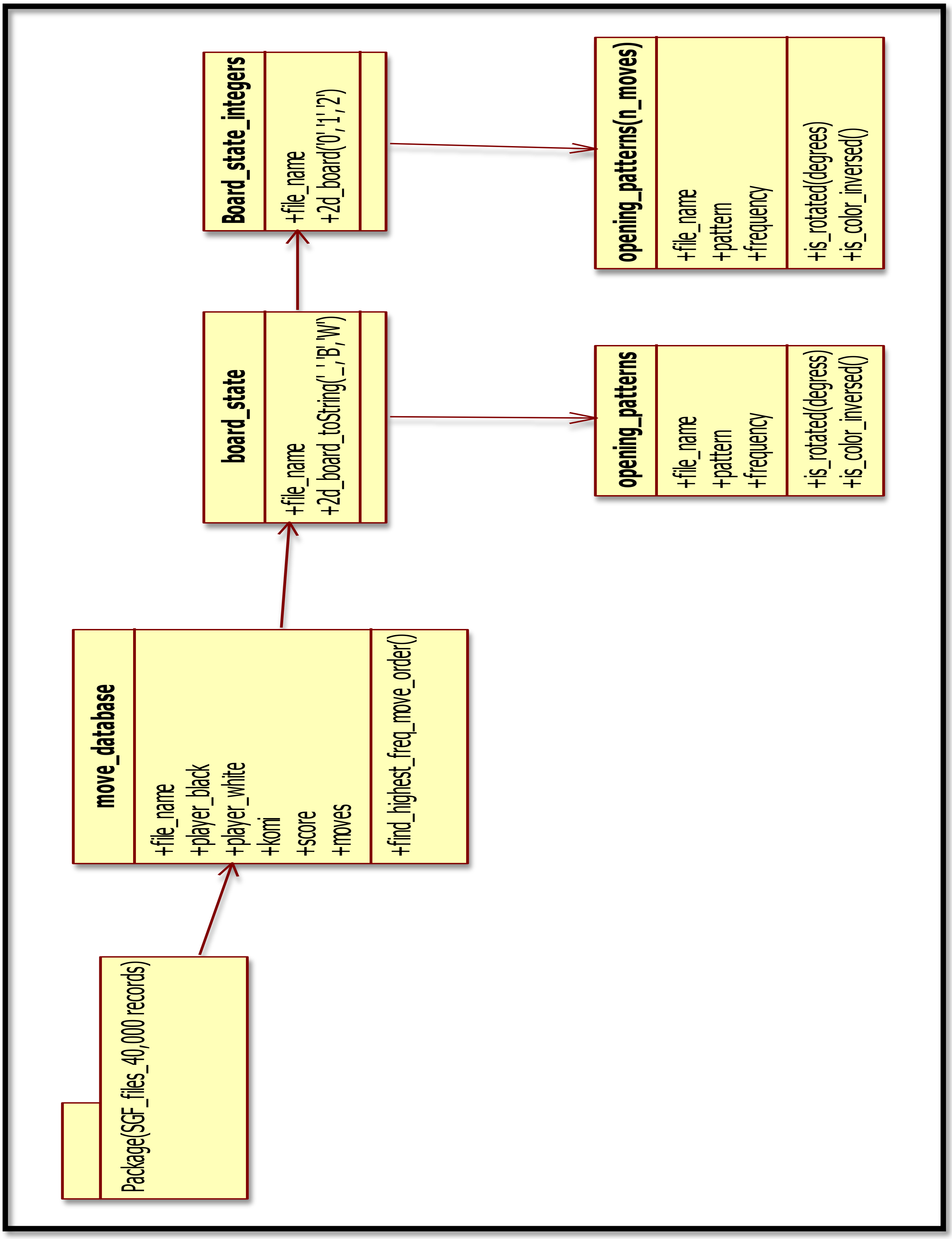


Fig 4. Class diagram

3.4. Modules

3.4.1. Reflection along x-axis.

```
char[][] reflection_x(char array[][])        // reflection along x axis
  {
     int row=array.length;
     int col=array[0].length;
     char[][] ret=new char[row][col];
     for(int i=0;i<row;i++)
     {  for(int j=0;j<col;j++)
        {  ret[i][j]=array[row-1-i][j];
        }
     }
     return ret;
  }
```

3.4.2. Reflection along y axis

```
  char[][] reflection_y(char array[][])                        // reflection along y axis
  {
     int row=array.length;
     int col=array[0].length;
     char[][] ret=new char[row][col];
      for(int i=0;i<row;i++)
     {  for(int j=0;j<col;j++)
        {  ret[i][j]=array[i][col-1-j];
        }
     }
     return ret;
  }
```

3.4.3. Reflection along origin

```
  char[][] reflection_origin(char array[][])                        // reflection along origin
  {        int row=array.length;
          int col=array[0].length;
         char[][] ret=new char[row][col];
      for(int i=0;i<row;i++)
     {  for(int j=0;j<col;j++)
        {   ret[i][j]=array[row-1-i][col-1-j];
        }
      }
       return ret;
    }
```

3.4.4. **Flipping of colors.**

```
char[][] inverse( char array[][])                              // flipping the colors
  {   int row, col;
      int i=array.length;
      int j=array[0].length;
      char [][] array_inverse=new char[i][j];
      for(row=0;row<i;row++)
         {  for(col=0;col<j;col++)
             {  if(array[row][col]=='B')array_inverse[row][col]='W';
                else if(array[row][col]=='W')array_inverse[row][col]='B';
                else array_inverse[row][col]='_';
             }
          }
      return array_inverse;
  }
```

3.4.5. **Rotation of the array through 90 degrees clockwise**

```
 public char[][] rotate(char[][] mat)
  {
  final int M = mat.length;
  final int N = mat[0].length;
  char[][] ret = new char[M][N];
  for (int r = 0; r < M; r++)
  { for (int c = 0; c < N; c++)
      {  ret[c][N-1-r] = mat[r][c];
      }
  }
  return ret;
}
```

3.4.6. Finding the patterns in the shape of a diamond.

```
char[][] k_pattern(int k, char array[][])
  {
     int size=(2*k)+1;
     char ret[][]=new char[size][size];
     for(int i=0;i<size;i++)
     {
        for(int j=0;j<size;j++)
        {
           ret[i][j]=' ';
        }
     }

      for(int i=0;i<=k;i++)                    //upper part of pattern
     {
        for(int j=k-i;j<=k+i;j++)
        {
           ret[i][j]=array[i][j];
        }
     }
     for(int i=k+1;i<size;i++)                //lower part of pattern
     {
        for(int j=i-k;j<size-i+k;j++)
        {
           ret[i][j]=array[i][j];
        }
     }
     return ret;
  }
```

4. Implementation

4.1. Tools used

4.1.1. General introduction

Since the project is a simple patterns generation module and there are no current tools or APIs available for the game of Go, I had decided to write my own scripts: to read the folders and its individual files, extracts the moves in order, create sample board models and then to generate patterns of varying sizes and shapes and find out the frequency of not only the general patterns but also of the opening "JOSEKI" board states.

For coding in JAVA and MySQL	-	**'Net beans IDE 7.0.1'**
For viewing the SGF files	-	**'GoGUI'.**

4.1.2. Reason for selecting tool

JAVA as a language was used since it is an OOPs language. Since in this project a lot of modules/ functions had to be reused in other .java files the language was convenient.

MySQL is an open source database and can be easily integrated in the Net beans IDE and thus was used.

GoGUI was the only freely available suite which could be used to view the SGF files and its additional functionality was an added advantage. Scoring and territory marking are a few functions available in it. Plus the code is freely available and thus can be configured to suit the project's needs.

4.2. Implementation

4.2.1. Basic methodology

* Collect records of professional go players.

* Use the records in SGF format to see the results in GOGUI software, for analysis.

* Use JAVA and MySQL to read the SGF files.

* Use players' information as the primary key for table and store the move text in a column

* Use the database records and find patterns in the data, which can be used for predicting the type of attack. Remove its canonical forms by checking for symmetry by rotation, reflection and colour flips.

* Pattern frequency generation program is used to get the "FUSEKI" and "JOSEKI" patterns.

* To select the next best move by the computer, use the pattern analysed result to filter the best local move available

* find the don’t care patterns and remove its canonical forms

4.2.2. Pseudo-codes and their outputs

Table 1 Pseudo codes and outputs

PSEUDOCODE

finding and adding the common fuseki patterns in db.

Fuseki is the whole board opening in the game of Go.

Steps:

```
•  open database connection
•  String sql ←"select file_name, move_made, result from
   move ";
•  ResultSet rs ← st.executeQuery(sql);
•  While(rs.next)
      do
   Get ‘file_name’, move_made’, ‘result’.
•  To find the fuseki patterns we find the first 10 moves
   from all the game records. i.e. first 60 characters.
•  Then we check all the moves in the order that it was
   played in.
•    for(int j=0;j<counter;j++)
   do
        count← 0
  for(int i=j+1;i<counter;i++)  do

 if((copy[j]!=null) && (copy[j].equals(copy[i])))

do
       copy[i]← null;
       count++;

 end for
```

SAMPLE OUTPUT

SAMPLE OUTPUT:

- Database: gouser
- Table : fuseki

```
mysql> select * from fuseki where
frequency >100;
```

Pattern	Frequency
;B[pd];W[dd];B[qp];W[dq];B[do];W[co];B[dp];W[cp];B[eq];W[cn]	104

```
if(copy[j]!=null && count!=0 )

do

      String sql_1="insert into fuseki(pattern,frequency ) "
       + "values(\""+copy[j]+"\""
              +",\""+count+"\")";
```

- Create statement for insert
- Execute statement

* End for
* End main.

Finding the duplicates in the fuseki patterns. (paper_pattern.java)

AIM:

- the purpose of this program is to not take the order of the moves into consideration
- only to understand that the board state with the positions are important to generate patterns, not the order of the moves.

PURPOSE:

- in ‘fuseki_find_pattern.java’ i found the (ordered_move_pattern, frequency) and stored it in the database ‘Fuseki.db’.
- now I will extract the patterns from the table and find the board state pattern from this and calculate the frequency of the patterns and their canonical forms.
- Canonical form set of a particular pattern is the set which is formed by the rotation, reflection and subsequent flipping of colours of the patterns in the set.
- The first original pattern set got by extraction from the .sgf files are then mapped and compared to its canonical pattern set by immediate computation

- Only the original pattern is saved
- This avoids duplicity.

STEPS:

- open database connection
- sql_select←"select pattern from fuseki";
- ResultSet rs ← st.executeQuery(sql);
- While(rs.next)

```
 do
Get ‘file_name’, move_made’,  ‘result’.
```

- To find the fuseki patterns we find the first 10 moves from all the game records. i.e. first 60 characters.
- Then we check all the moves in the order that it was played in.
- for(int j=0;j<counter;j++)

```
do
        count← 0
   for(int i=j+1;i<counter;i++)  do

  if((copy[j]!=null) &&    (copy[j].equals(copy[i])))

do
        copy[i]← null;
        count++;

  end for

if(copy[j]!=null && count!=0 )

do

     String sql_1="insert into fuseki(pattern,frequency ) "
      + "values(\""+copy[j]+"\""
             +",\""+count+"\")";
```

- Create statement for insert

- Execute statement
- End for

End main.

Finding the 3X3 patterns

```
for(int loop=0;loop<counter;loop++)
{
  //2d array created
    int row, col;
    for(row=0;row<19;row++)  // copy files in a 2d array
    {
        for(col=0;col<19;col++)
        {
            array[row][col]=move_made_ch[loop][c];
            c++;
        }
    }
//for each file
    for(row=0;row<19-3;row++) // for all rows
    {
        for(col=0;col<19-3;col++) // for all columns
        {
//creating a 3x3 pattern
            for(int i=0;i<3;i++)
            {
                for(j=0;j<3;j++)
                {
                   pattern_array[i][j]=array[i+row][j+col];
// first pattern --> the original pattern
                 }
            }
            String str=new String();
            str=obj.charToString(pattern_array);
            String empty="___________________________";
            if(str.equals(empty))
            {
```

```
pattern no...=>  2409
...............original.........................
_ _ _
B _ W
B B B
....................................................
.
................inverse.......................
_ _ _
W _ B
W W W
....................................................
...............original_90...................
B B _
B _ _
B W _
...............original_90_inverse.......

W W _
W _ _
W B _
....................................................

...............original_180..................
B B B
W _ B
_ _ _
....................................................
.
...............original_180_inv............
W W W
B _ W
```

```
                    continue;
          }



          //check if pattern matches some existing string
already presesnt in db else write in db
          //1. convert all the patterns into string and check db
          //2. use the select statement to find a match
          //3. if no match update db with the original pattern
          //4. else add the frequency+1
```

- **Find all the canonical forms of the original patterns by rotation**
- **Reflection**
- **Flipping of colors**

```
if((st_pattern_array.equals(st_pattern_array_360))&&
(st_pattern_array_inverse.equals(st_pattern_array_inverse_3
60)))
               System.out.println("CORRECT!!!!!"+loop);
             else
System.out.println("NOOOOOOOOOOOOO!!!!!");

     */
          for(int l=0;l<d;l++)
          {

                    if((pattern_db_array[l].equals("any of the
canonical forms"))
             {
               //  System.out.println("****match found at*****
"+l);
               flag=1; frequency[l]++;break;
             }
           }

     if(flag==0)
```

```
_ _ _
.....................................................
.

...............original_270...................

_ W B
_ _ B
_ B B
.....................................................
...............original_270_inv...........

_ B W
_ _ W
_ W W
.....................................................
...............original_x.......................
B B B
B _ W
_ _ _
.....................................................
...............original_x_inv...............
W W W
W _ B
_ _ _
.....................................................
...............original_y.....................
_ _ _
W _ B
B B B
.....................................................
...............original_y_inv................

_ _ _
B _ W
W W W
```

```
      {
pattern_db_array[d]=obj.charToString(pattern_array);
         frequency[d]++;
         d++;
      }   }} }
```

Finding the diamond shaped patterns- K-nearest pattern

```
  char[][] k_pattern(int k, char array[][])
  {
     int size=(2*k)+1;
     // since the middlemost row and col length=2k+1
     //the rows above and below it decreases by 2 on both
directions
     char ret[][]=new char[size][size];
     for(int i=0;i<size;i++)
     {
        for(int j=0;j<size;j++)
        {
           ret[i][j]=' ';
        }
     }
      // dynamically create array of 2k+1 by 2k+1

      for(int i=0;i<=k;i++) //upper part of pattern
     {
        for(int j=k-i;j<=k+i;j++)
        {
           ret[i][j]=array[i][j];
        }
     }
     for(int i=k+1;i<size;i++) //lower part of pattern
     {
        for(int j=i-k;j<size-i+k;j++)
        {
           ret[i][j]=array[i][j];
```

```
 [7142]...
    _
  _ _ _
_ _ B B _
  _ B W
    W
Frequency=8
K bits of info=5

 [9794]...
      B
    _ B B
B B B W W
   W B B
      W

Frequency=6
K bits of info=12

[5198]...

     W
   W B W
B W B B B
   B B _
     _
Frequency=37
K bits of info=11
```

}
}
return ret;
}
	[5202]...
	B
	B B B
	W W B _ _
	W W B
	B
	Frequency=36
	K bits of info=11

Reducing the patterns using don’t care conditions
 	
for(int i=0;i<counter;i++) // for each pattern(2d array)
{
 	
//for each position of the template from 0-8..
//adda dont care condition and then check the other
// patterns and its varients for a match.
// if for a particular don't care
// ‘B’, 'W' and '_' all are presesnt store the pattern template
//else stores the whole array.
 	
for(int j=0;j<9;j++)
{
 	
flag=0;
pattern_template=st_pattern_array[i].toCharArray();
 	
pattern_template[j]='B';
 	
st_pattern_template_B=obj.oneDtoString(pattern_template);
 	
pattern_template[j]='W';
	.....[10729]......
	
	_ W _
	_ W _
	B W ?
	
	.....[10730]......
	
	_ ? _
	_ W _
	B W W
	
	.....[10731]......
	_ W _
	? W _
	B W W
	
	.....[10732]......
	_ W _
	_ W _
	? W W
	
	.....[10733]......
	_ W _
	_ W _

```
st_pattern_template_W=obj.oneDtoString(pattern_template)
;

         pattern_template[j]='_';

st_pattern_template_blank=obj.oneDtoString(pattern_templ
ate);

         pattern_template[j]='?'; //dont care template

st_pattern_template_temp=obj.oneDtoString(pattern_templa
te);

         boolean x,y,z;

        for(int k=0;k<counter;k++)
        {
        x=(st_pattern_template_B.equals("all canonical form
of any pattern"))
if(x)
{
       for(int k1=0;k1<counter;k1++)
    {
        y=(st_pattern_template_B.equals("all canonical form
of any pattern"))        if(y)
{

    for(int k2=0;k2<counter;k2++)
    {
        z=(st_pattern_template_B.equals("all canonical form
of any pattern"))
                    if(z)
                    {
                       // a pattern with "b", "w" and "_" all found
                        flag=1;
                        break;
```

```
B ? W

.....[10734]......
? W _
_ W _
B _ B

.....[10735]......
_ ? _
_ W _
B _ B
```

``` } } } } } } ```	
```
if(flag==1)
{
    pattern_3by3_final[d]=st_pattern_template_temp;
    flag=0;
    d++;
}
}
}
```

4.3. Testing

4.3.1. Test approach

A modular testing approach would be used through this experiment, wherein the modules of reflection, rotation and flipping would be separately checked.

Secondly, the patterns generated would be checked if they are formed okay or not.

4.3.2. Test plan

4.3.2.1. Features to be tested

- Rotation
- Reflection
- Flipping of colors
- Board formation
- 3X3 patterns
- 5X5 patterns
- K_nearest (diamond shaped) patterns

4.3.2.2. Features not to be tested

- The frequency of all the patterns would not be checked.

4.3.3. Testing tools and environment

Manual testing would be done , since we already know the sample output vs input.

4.3.4. Test cases

4.3.4.1. Case –n

- Rotation

- Reflection
- Flipping of colors
- Board formation
- 3X3 patterns
- 5X5 patterns
- K_nearest (diamond shaped) patterns

4.3.4.2. Purpose

• Rotation	This function rotates the NxN array by 90 degrees
• Reflection	This function reflects the NxN array along x and y axis
• Flipping of colors	This function interchanges the colour of the NxN array
• Board formation	This function forms 19x19 boards
• 3X3 patterns	This function creates 3x3 square patterns
• 5X5 patterns	This function creates 5x5 square patterns
• K_nearest (diamond shaped) patterns	This function creates the 'K info bit ' diamond pattern

Table 2. purpose of test cases

4.3.4.3. Inputs

• Rotation		Original	_ _ _ B _ W B B B	
		Original_90	B B _ B _ _ B W _	
		Original_180	B B B W _ B _ _ _	
		Original_270	_ W B _ _ B _ B B	
		Original_360	_ _ _ B _ W B B B	
• Reflection		Original	_ _ _	

	B _ W B B B
• Flipping of colors	Original | _ _ _ B _ W B B B _ _ _ W _ B B B B
• Board formation	MOVE====>;B[qd];W[dc];B[dp];W[pq];B[ce];W[oc];B[ld];W[of];B[oe]; ……………………..)
• 3X3 patterns	_ W W B _ _ _ _ _ _ _ _ _ _ B B W _ _ _ W B B _ _ _ B B B _ B B B B W W _ _ _ _ W B B _ B W B B B W W B W B W _ _ W W W B W B _ B W B W W B W W B W _ _ _ B W W W W B W W W W _ W B W B B W _ B B W B W _ B B W _ W W W B W _ B W _ B W W B W W B W B B W B W B B B B W _ _ B B W W W W B B W W W W B W B W W W _ _ B W _ W B B B B _ _ B B W W W _ B B _ B W _ W W B W B B B B _ B _ B B _ _ _ B W _ W B B B W B W B B B _ B _ _ B _ B B W B W W W W W W W W B W W W W W B B W W B W B _ W B B W _ W B W _ _ W W W W W B W B W W B W B _ _ W B W _ _ _ W B B B B B B W B W W W W W B W W _ W B B B B B _ B B B B W W B B W W B W W W B B W W W B B B B B W B _ B B B B W B B B W _ W B W W W B W B _ B _ _ B B B B W B B B W W W W W W W B _ _ _
• 5X5 patterns	_ W W B _ _ _ _ _ _ _ _ _ _ B B W _ _ _ W B B _ _ _ B B B _ B B B B W W _ _ _ _ W B B _ B W B B B W W B W B W _ _ W W W B W B _ B W B W W B W W B W _ _ _ B W W W W B W W W W _ W B W B B W _ B B W B W _ B B W _ W W W B W _ B W _ B W W B W W B W B B W B W B B B B W _ _ B B W W W W B B W W W W B W B W W W _ _ B W _ W B B B B _ _ B B W W W _ B

	B _ B W _ W W B W B B B B _ B _ B B _ _ _ B W _ W B B B W B W B B B _ B _ _ B _ B B W B W W W W W W W W B W W W W W B B W W B W B _ W B B W _ W B W _ _ W W W W W B W B W W B W B _ _ W B W _ _ _ W B B B B B B W B W W W W W B W W _ W B B B B B _ B B B B W W B B W W B W W W B B W W W B B B B B W B _ B B B B W B B B W _ W B W W W B W B _ B _ _ B B B B W B B B W W W W W W W B _ _ _
• K_nearest (diamond shaped) patterns	[9794]... B _ B B B B B W W W B B W Frequency=6 K bits of info=12

Table 3. table of inputs of test cases

4.3.4.4. Expected outputs and Pass/ fail criteria

• Rotation	Original _ _ _ B _ W B B B		90deg B B _ B _ _ B W _		**Pass**
			180 deg B B B W _ B _ _ _		
			270 deg _ W B _ _ B _ B B		
			360 deg _ _ _		

		B _ W B B B	
• Reflection	Original _ _ _ B _ W B B B	...............original_x....................... B B B B _ W _ _ _ ....original_y..................... _ _ _ W _ B B B B	**Pass**
• Flipping of colors	Original _ _ _ B _ W B B B	Original_inverse _ _ _ W _ B WWW	**Pass**
• Board formation	**1.sgf**	_ W W B _ _ _ _ _ _ _ _ _ _ B B W _ _ _ W B B _ _ _ B B B _ B B B B W W _ _ _ _ W B B _ B W B B B W W B W B W _ _ W W W B W B _ B W B W W B W W B W _ _ _ B W W W W B W W W W _ W B W B B W _ B B W B W _ B B W _ W W W B W _ B W _ B W W B W W B W B B W B W B B B B W _ _ B B W W W W B B W W W W B W B W W W _ _ B W _ W B B B B _ _ B B W W W _ B B _ B W _ W W B W B B B B _ B _ B B _ _ _ B W _ W B B B W B W B B B _ B _ _ B _ B B W B W W W W W W W W B W W W W W B B W W B W B _ W B B W _ W B W _ _ W W W W W B W B W W B W B _ _ W B W _ _ _ W B B B B B B W B W W W W W B W W _ W B B B B B _ B B B B W W B B W W B W W W B B W W W B B B B B W B _ B B B B W B B B W _ W B W W W B W B _ B _ _ B B B B W B B B W W W W W W W B _ _ _	**Pass**
• 3X3 patterns	**Record**	_ _ _ B _ W B B B	**Pass**
• 5X5 patterns	**Record**	__WBW WWWBW WBBWW WB_BB	**Pass**

		WB_B_	
• K_nearest(diamond) patterns	[9794]... B _ B B B B B W W W B B W	B _ B B B B B W W W B B W Frequency=6 K bits of info=12	**Pass**

Table 4. final test analysis

5. Results and discussion

5.1. Results

The experiments on “pattern generation using Go game records” allow making a few conclusion remarks at the end of the paper.

First, I can say that to extract patterns the stones’ spatial positions on the board relative to the others are required. The order of the moves made could be irrelevant to the pattern altogether.

Secondly, symmetry properties must be removed in order to effectively obtain statistical usages of patterns, otherwise, the number of patterns will increase significantly and related patterns cannot be classified together.

Thirdly, there are a large amount of quality patterns that can be extracted from professional game records. In this project, I have generated codes for 3x3, 5x5 square and diamond patterns, but with a few minor changes the sizes of the pattern array can be changed to get more variety as per the program’s need.

Fourthly, statistical usages of patterns indicate relative usefulness and urgency of patterns, which can be used to help improve the strength of computer programs. Apart from this in the “K_means_paperalgo.java”, k_bits of information along with the frequency is used to create some sort of ordering. Lower frequency and lower k information bit patterns could be discarded, in case of space constraints.

5.2. Performance Analysis

With the whole process broken down into smaller modules there is no memory and processor constraints. The largest script takes about 10- 20 minutes to execute. Moreover, the time complexity of the script is proportional to the size of the game records, so for the previous testing phase, 100 records are checked and then finally, all the records are used for generation.

The number of patterns generated originally follows the **equation=’ (19-n) x (19-n)’** where n is the size of the pattern to be generated. After the canonical forms on the basis of symmetry are removed, the patterns are radically reduced. The program therefore is SCALABLE, i.e. any number of records can be added and the script need not be changed to incorporate the changes to the record number.

6. Conclusion and future enhancements

In this project, I have generated different sizes of pattern with two primary shapes, square and diamond. For the square shapes, the frequency of the patterns is also found out. For the diamond shape, the K- bits of information is found out which tells us that, out of the 13 spaces of the pattern template, how many have been occupied. These two attributes could be used to form clusters, using the WEKA tool.

The symmetrical patterns have been removed so that unique patterns could be added to the database. When and however the need arises, the rotation, reflection and the color switch functions could be used to find the pattern match.

GNU Go has a set of pattern templates wherein the similar patterns are grouped together and there are positions marked as don't care positions. These don't care positions are areas on the patterns where it doesn't matter whether 'white' or 'black' plays or it remains empty, the rules would be followed nonetheless.

Similarly, on those lines I thought of creating pattern templates automatically, since my patterns are also computer generated. For the template creation, I first created a blank template and checked for all positions whether or not the template function fir it or not. If yes, the template was approved and checked for its canonical forms. If none matched in the database already created, then include it in the database else skip.

For the future, I would move towards creating an algorithm wherein the patterns could be classified according to the play type they are best suited for. There are various classification algorithms which could be used.

7. REFERENCES

[1] *Documentation about SGF format*. http://www.red-bean.com/sgf/

[2] *American Go Association*. http://www.usgo.org.

[3] *GNU go documentation*, http://www.gnu.org/software/gnugo/gnugo_toc.html

[4] Brent Harrison, David L. Roberts, *Using Sequential Observations to Model and Predict Player Behaviour*.

[5] Sylvain Gelly, Yizao Wang , R´emi Munos , Olivier Teytaud, *Modification of UCT with Patterns in Monte-Carlo Go*, 2006

[6] LIU Zhi-qing, DOU Qing, Automatic *pattern acquisition from game records in go*, THE JOURNAL OF CHINA UNIVERSITIES OF POSTS AND TELECOMMUNICATIONS, Volume 13, Issue 4, December 2006

[7] T. Cazenave. *Generation of patterns with external conditions for the game of GO.*

[8] Sylvain Gelly, David Silver, *Monte-Carlo tree search and rapid action value estimation in computer Go*

[9] Woo-Jun PARK, *Representation and Comparison of Fuseki Patterns of Go-games*

[10] Emil H.J., *Learning Patterns in the Game of Go* , MSc Thesis

[11] Martin M¨uller. *Computer go. Artificial Intelligence*, 134(1):145–179, 2002.

[12] Brett Alexander Harrison, *Move Prediction in the Game of Go*, thesis for Harvard College

Cambridge, Massachusetts,April 1, 2010

[13] Wei Xin, Sun Yinglong, Yang Hui, Wang Jiao(Corresponding Author), *The Research of Pattern Symmetry Problem in Learning in Computer Go*, IEEE 2012.

[14] Nakamura T., *.Acquisition of move sequence patterns from game record database using n-gram statistics.*, In Game Programming Workshop 97, 1997.

8. APPENDIX- I

8.1. PATTERNS TEMPLATES FOUND for number11512

```
.....[0]......
? B B
B B B
B B B

.....[1]......
B ? B
B B B
B B B

.....[2]......
B B B
B ? B
B B B

.....[3]......
? B B
B B B
B B W

.....[4]......
B ? B
B B B
B B W

.....[5]......
B B ?
B B B
B B W

.....[6]......
B B B
? B B
B B W

.....[7]......
B B B
B ? B
B B W

.....[8]......
B B B
B B ?
B B W

.....[9]......
B B B
B B B
? B W

.....[10]......
B B B
B B B
B ? W

.....[11]......
? B B
B B B
W W W

.....[12]......
B ? B
B B B
W W W

.....[13]......
B B B
? B B
W W W

.....[14]......
B B B
B ? B
W W W

.....[15]......
B B B
B B B
? W W

.....[16]......
B B B
B B B
W ? W

.....[17]......
? B B
B B W
B B W

.....[18]......
B ? B
B B W
B B W

.....[19]......
B B ?
B B W
B B W

.....[20]......
B B B
? B W
B B W

.....[21]......
B B B
B ? W
B B W

.....[22]......
B B B
B B W
? B W

.....[23]......
B B B
B B W
B ? W

.....[24]......
B B B
B B W
B B ?

.....[25]......
? B B
B B W
W W _

.....[26]......
B ? B
B B W
W W _

.....[27]......
B B ?
B B W
W W _

.....[28]......
B B B
? B W
W W _

.....[29]......
B B B
B ? W
W W _

.....[30]......
B B B
B B ?
W W _

.....[31]......
B B B
B B W
? W _

.....[32]......
B B B
B B W
W ? _

.....[33]......
B B B
B B W
W W ?

.....[34]......
? B B
B B W
W _ W

.....[35]......
B ? B
B B W
W _ W

.....[36]......
B B ?
B B W
W _ W

.....[37]......
B B B
? B W
W _ W

.....[38]......
B B B
B ? W
W _ W

.....[39]......
B B B
B B ?
W _ W

.....[40]......
B B B
B B W
? _ W

.....[41]......
B B B
B B W
W ? W

.....[42]......
B B B
B B W
W _ ?

.....[43]......
? B B
B B W
_ _ W

.....[44]......
B ? B
B B W
_ _ W

.....[45]......
B B ?
B B W
_ _ W

.....[46]......
B B B
? B W
_ _ W

.....[47]......
B B B
B ? W
_ _ W

.....[48]......
B B B
B B ?
_ _ W

.....[49]......
B B B
B B W
_ ? W

.....[50]......
B B B
B B W
_ _ ?

.....[51]......
? B B
B B _
B W B

.....[52]......
B ? B
B B _
B W B

.....[53]......
B B ?
B B _
B W B

.....[54]......
B B B
? B _
B W B

.....[55]......
B B B
B ? _
B W B

.....[56]......
B B B
B B ?
B W B

.....[57]......
B B B
B B _
? W B

.....[58]......
B B B
B B _
B ? B

.....[59]......
B B B
B B _
B W ?

.....[60]......
? B B
B B _
B _ _

.....[61]......
B ? B
B B _
B _ _

.....[62]......
B B ?
B B _
B _ _

.....[63]......
B B B
? B _
B _ _

.....[64]......
B B B
B ? _
B _ _

.....[65]......
B B B
B B ?
B _ _

.....[66]......
B B B
B B _
? _ _

.....[67]......
B B B
B B _
B ? _

.....[68]......
B B B
B B _
B _ ?

.....[69]......
? B B
B B _
W B W

.....[70]......
B ? B
B B _
W B W

.....[71]......
B B ?
B B _
W B W

.....[72]......
B B B
? B _
W B W

.....[73]......
B B B
B ? _
W B W

.....[74]......
B B B
B B ?
W B W

.....[75]......
B B B
B B _
? B W

.....[76]......
B B B
B B _
W ? W

.....[77]......
B B B
B B _
W B ?

.....[78]......
? B B
B B _
W W W

.....[79]......
B ? B
B B _
W W W

.....[80]......
B B ?
B B _
W W W

.....[81]......
B B B
? B _
W W W

.....[82]......
B B B
B ? _
W W W

.....[83]......
B B B
B B _
? W W

.....[84]......
B B B
B B _
W W ?

.....[85]......
? B B
B W B
B B W

.....[86]......
B ? B
B W B
B B W

.....[87]......
B B ?
B W B
B B W

.....[88]......
B B B
? W B
B B W

.....[89]......
B B B
B W ?
B B W

.....[90]......
B B B
B W B
? B W

.....[91]......
B B B
B W B
B ? W

.....[92]......
B B B
B W B
B B ?

.....[93]......
? B B
B W B
W W _

.....[94]......
B ? B
B W B
W W _

.....[95]......
B B ?
B W B
W W _

.....[96]......
B B B
? W B
W W _

.....[97]......
B B B
B ? B
W W _

.....[98]......
B B B
B W ?
W W _

.....[99]......
B B B
B W B
? W _

.....[100]......
B B B
B W B
W ? _

.....[101]......
B B B
B W B
W W ?

.....[102]......
? B B
B W B
W _ W

.....[103]......
B ? B
B W B
W _ W

.....[104]......
B B B
? W B
W _ W

.....[105]......
B B B
B ? B
W _ W

.....[106]......
B B B
B W B
? _ W

.....[107]......
B B B
B W B
W ? W

.....[108]......
? B B
B W B
_ W B

.....[109]......
B ? B
B W B
_ W B

.....[110]......
B B ?
B W B
_ W B

.....[111]......
B B B
? W B
_ W B

.....[112]......
B B B
B ? B
_ W B

.....[113]......
B B B
B W ?
_ W B

.....[114]......
B B B
B W B
? W B

.....[115]......
B B B
B W B
_ ? B

.....[116]......
? B B
B W _
B W _

.....[117]......
B ? B
B W _
B W _

.....[118]......
B B ?
B W _
B W _

.....[119]......
B B B
? W _
B W _

.....[120]......
B B B
B ? _
B W _

.....[121]......
B B B
B W _
? W _
```

```
.....[122]......  .....[135]......  .....[148]......  .....[161]......  .....[174]......  .....[187]......  .....[200]......  .....[213]......  .....[226]......  .....[239]......
     B B B             B B ?             B B B             B B ?             B B B             B B B             B ? B             B B B             B B B             B B B
     B W _             B W _             B _ B             B _ W             B _ W             W ? B             W B W             W B W             ? B _             W W B
     B ? _             _ W B             _ ? B             W _ W             _ ? B             B W B             B W W             B ? B             W _ W             W B ?

.....[123]......  .....[136]......  .....[149]......  .....[162]......  .....[175]......  .....[188]......  .....[201]......  .....[214]......  .....[227]......  .....[240]......
     B B B             B B B             B B B             B B B             B B B             B B B             B B ?             ? B B             B B B             ? B B
     B W _             ? W _             B _ B             ? _ W             B _ W             W B ?             W B W             W B W             W ? _             W W B
     B W ?             _ W B             _ W ?             W _ W             _ _ ?             B W B             B W W             W W W             W _ W             W _ W

.....[124]......  .....[137]......  .....[150]......  .....[163]......  .....[176]......  .....[189]......  .....[202]......  .....[215]......  .....[228]......  .....[241]......
     ? B B             B B B             ? B B             B B B             ? B B             B B B             B B B             B ? B             B B B             B ? B
     B W _             B ? _             B _ W             B _ ?             B _ _             W B B             ? B W             W B W             W B _             W W B
     _ B B             _ W B             W B B             W _ W             W B B             ? W B             B W W             W W W             ? _ W             W _ W

.....[125]......  .....[138]......  .....[151]......  .....[164]......  .....[177]......  .....[190]......  .....[203]......  .....[216]......  .....[229]......  .....[242]......
     B ? B             B B B             B ? B             B B B             B ? B             B B B             B B B             B B B             B B B             B B ?
     B W _             B W _             B _ W             B _ W             B _ _             W B B             W ? W             ? B W             W B _             W W B
     _ B B             ? W B             W B B             ? _ W             W B B             B W ?             B W W             W W W             W ? W             W _ W

.....[126]......  .....[139]......  .....[152]......  .....[165]......  .....[178]......  .....[191]......  .....[204]......  .....[217]......  .....[230]......  .....[243]......
     B B ?             B B B             B B ?             B B B             B B ?             ? B B             B B B             B B B             B B B             B B B
     B W _             B W _             B _ W             B _ W             B _ _             W B B             W B ?             W ? W             W B _             W W ?
     _ B B             _ W ?             W B B             W ? W             W B B             _ W _             B W W             W W W             W _ ?             W _ W

.....[127]......  .....[140]......  .....[153]......  .....[166]......  .....[179]......  .....[192]......  .....[205]......  .....[218]......  .....[231]......  .....[244]......
     B B B             ? B B             B B B             B B B             B B B             B ? B             B B B             B B B             ? B B             B B B
     ? W _             B _ B             ? _ W             B _ W             ? _ _             W B B             W B W             W B W             W W B             W W B
     _ B B             B B B             W B B             W _ ?             W B B             _ W _             ? W W             W ? W             W B W             ? _ W

.....[128]......  .....[141]......  .....[154]......  .....[167]......  .....[180]......  .....[193]......  .....[206]......  .....[219]......  .....[232]......  .....[245]......
     B B B             B ? B             B B B             ? B B             B B B             B B ?             B B B             ? B B             B ? B             B B B
     B ? _             B _ B             B ? W             B _ W             B ? _             W B B             W B W             W B W             W W B             W W B
     _ B B             B B B             W B B             _ _ B             W B B             _ W _             B ? W             W _ W             W B W             W _ ?

.....[129]......  .....[142]......  .....[155]......  .....[168]......  .....[181]......  .....[194]......  .....[207]......  .....[220]......  .....[233]......  .....[246]......
     B B B             ? B B             B B B             B ? B             B B B             B B B             B B B             B ? B             B B ?             ? B B
     B W ?             B _ B             B _ ?             B _ W             B _ _             ? B B             W B W             W B W             W W B             W W B
     _ B B             _ W B             W B B             _ _ B             ? B B             _ W _             B W ?             W _ W             W B W             _ W _

.....[130]......  .....[143]......  .....[156]......  .....[169]......  .....[182]......  .....[195]......  .....[208]......  .....[221]......  .....[234]......  .....[247]......
     B B B             B ? B             B B B             B B ?             B B B             B B B             ? B B             B B B             B B B             B ? B
     B W _             B _ B             B _ W             B _ W             B _ _             W ? B             W B W             W ? W             ? W B             W W B
     ? B B             _ W B             ? B B             _ _ B             W ? B             _ W _             B _ B             W _ W             W B W             _ W _

.....[131]......  .....[144]......  .....[157]......  .....[170]......  .....[183]......  .....[196]......  .....[209]......  .....[222]......  .....[235]......  .....[248]......
     B B B             B B ?             B B B             B B B             B B B             B B B             B ? B             B B B             B B B             B B ?
     B W _             B _ B             B _ W             ? _ W             B _ _             W B ?             W B W             W B W             W ? B             W W B
     _ ? B             _ W B             W ? B             _ _ B             W B ?             _ W _             B _ B             ? _ W             W B W             _ W _

.....[132]......  .....[145]......  .....[158]......  .....[171]......  .....[184]......  .....[197]......  .....[210]......  .....[223]......  .....[236]......  .....[249]......
     B B B             B B B             B B B             B B B             ? B B             B B B             B B B             ? B B             B B B             B B B
     B W _             ? _ B             B _ W             B ? W             W B B             W B B             ? B W             W B _             W W ?             ? W B
     _ B ?             _ W B             W B ?             _ _ B             B W B             ? W _             B _ B             W _ W             W B W             _ W _

.....[133]......  .....[146]......  .....[159]......  .....[172]......  .....[185]......  .....[198]......  .....[211]......  .....[224]......  .....[237]......  .....[250]......
     ? B B             B B B             ? B B             B B B             B ? B             B B B             B B B             B ? B             B B B             B B B
     B W _             B _ ?             B _ W             B _ ?             W B B             W B B             W ? W             W B _             W W B             W W ?
     _ W B             _ W B             W _ W             _ _ B             B W B             _ ? _             B _ B             W _ W             ? B W             _ W _

.....[134]......  .....[147]......  .....[160]......  .....[173]......  .....[186]......  .....[199]......  .....[212]......  .....[225]......  .....[238]......  .....[251]......
     B ? B             B B B             B ? B             B B B             B B ?             ? B B             B B B             B B ?             B B B             B B B
     B W _             B _ B             B _ W             B _ W             W B B             W B W             W B W             W B _             W W B             W W B
     _ W B             ? W B             W _ W             ? _ B             B W B             B W W             ? _ B             W _ W             W ? W             ? W _
```

.....[252]......
B B B
W W B
_ ? _

.....[253]......
B B B
W W B
_ W ?

.....[254]......
? B B
W W W
W B _

.....[255]......
B ? B
W W W
W B _

.....[256]......
B B ?
W W W
W B _

.....[257]......
B B B
? W W
W B _

.....[258]......
B B B
W ? W
W B _

.....[259]......
B B B
W W ?
W B _

.....[260]......
B B B
W W W
? B _

.....[261]......
B B B
W W W
W ? _

.....[262]......
B B B
W W W
W B ?

.....[263]......
? B B
W W W
W _ B

.....[264]......
B ? B
W W W
W _ B

.....[265]......
B B ?
W W W
W _ B

.....[266]......
B B B
? W W
W _ B

.....[267]......
B B B
W ? W
W _ B

.....[268]......
B B B
W W ?
W _ B

.....[269]......
B B B
W W W
? _ B

.....[270]......
B B B
W W W
W ? B

.....[271]......
B B B
W W W
W _ ?

.....[272]......
? B B
W W W
_ B _

.....[273]......
B ? B
W W W
_ B _

.....[274]......
B B B
? W W
_ B _

.....[275]......
B B B
W ? W
_ B _

.....[276]......
B B B
W W W
_ ? _

.....[277]......
? B B
W W W
_ W _

.....[278]......
B ? B
W W W
_ W _

.....[279]......
B B B
W ? W
_ W _

.....[280]......
B B B
W W W
? W _

.....[281]......
? B B
W W _
_ _ W

.....[282]......
B ? B
W W _
_ _ W

.....[283]......
B B ?
W W _
_ _ W

.....[284]......
B B B
? W _
_ _ W

.....[285]......
B B B
W ? _
_ _ W

.....[286]......
B B B
W W ?
_ _ W

.....[287]......
B B B
W W _
? _ W

.....[288]......
B B B
W W _
_ ? W

.....[289]......
B B B
W W _
_ _ ?

.....[290]......
? B B
W _ B
B B _

.....[291]......
B ? B
W _ B
B B _

.....[292]......
B B ?
W _ B
B B _

.....[293]......
B B B
? _ B
B B _

.....[294]......
B B B
W ? B
B B _

.....[295]......
B B B
W _ ?
B B _

.....[296]......
B B B
W _ B
? B _

.....[297]......
? B B
W _ B
W _ _

.....[298]......
B ? B
W _ B
W _ _

.....[299]......
B B ?
W _ B
W _ _

.....[300]......
B B B
? _ B
W _ _

.....[301]......
B B B
W _ ?
W _ _

.....[302]......
B B B
W _ B
W ? _

.....[303]......
? B B
W _ B
_ B _

.....[304]......
B ? B
W _ B
_ B _

.....[305]......
B B ?
W _ B
_ B _

.....[306]......
B B B
? _ B
_ B _

.....[307]......
B B B
W ? B
_ B _

.....[308]......
B B B
W _ ?
_ B _

.....[309]......
B B B
W _ B
_ ? _

.....[310]......
B B B
W _ B
_ B ?

.....[311]......
? B B
W _ W
W B B

.....[312]......
B ? B
W _ W
W B B

.....[313]......
B B B
W ? W
W B B

.....[314]......
B B B
W _ ?
W B B

.....[315]......
B B B
W _ W
? B B

.....[316]......
B B B
W _ W
W ? B

.....[317]......
B B B
W _ W
W B ?

.....[318]......
? B B
W _ W
_ B _

.....[319]......
B ? B
W _ W
_ B _

.....[320]......
B B B
W _ W
? B _

.....[321]......
B B B
W _ W
_ ? _

.....[322]......
? B B
W _ W
_ _ B

.....[323]......
B ? B
W _ W
_ _ B

.....[324]......
B B ?
W _ W
_ _ B

.....[325]......
B B B
W ? W
_ _ B

.....[326]......
B B B
W _ ?
_ _ B

.....[327]......
B B B
W _ W
? _ B

.....[328]......
B B B
W _ W
_ ? B

.....[329]......
B B B
W _ W
_ _ ?

.....[330]......
? B B
W _ W
_ _ _

.....[331]......
B ? B
W _ W
_ _ _

.....[332]......
B B B
? _ W
_ _ _

.....[333]......
B B B
W ? W
_ _ _

.....[334]......
? B B
W _ _
W _ _

.....[335]......
B ? B
W _ _
W _ _

.....[336]......
B B ?
W _ _
W _ _

.....[337]......
B B B
? _ _
W _ _

.....[338]......
B B B
W ? _
W _ _

.....[339]......
B B B
W _ _
? _ _

.....[340]......
B B B
W _ _
W ? _

.....[341]......
B B B
W _ _
W _ ?

.....[342]......
? B B
W _ _
_ B W

.....[343]......
B ? B
W _ _
_ B W

.....[344]......
B B ?
W _ _
_ B W

.....[345]......
B B B
? _ _
_ B W

.....[346]......
B B B
W ? _
_ B W

.....[347]......
B B B
W _ ?
_ B W

.....[348]......
B B B
W _ _
? B W

.....[349]......
B B B
W _ _
_ ? W

.....[350]......
B B B
W _ _
_ B ?

.....[351]......
? B B
W _ _
_ W _

.....[352]......
B ? B
W _ _
_ W _

.....[353]......
B B ?
W _ _
_ W _

.....[354]......
B B B
? _ _
_ W _

.....[355]......
B B B
W ? _
_ W _

.....[356]......
B B B
W _ ?
_ W _

.....[357]......
B B B
W _ _
? W _

.....[358]......
B B B
W _ _
_ ? _

.....[359]......
B B B
W _ _
_ W ?

.....[360]......
? B B
_ B B
B B W

.....[361]......
B ? B
_ B B
B B W

.....[362]......
B B ?
_ B B
B B W

.....[363]......
B B B
_ B ?
B B W

.....[364]......
B B B
_ B B
B ? W

.....[365]......
B B B
_ B B
B B ?

.....[366]......
? B B
_ B B
B W W

.....[367]......
B ? B
_ B B
B W W

.....[368]......
B B ?
_ B B
B W W

.....[369]......
B B B
? B B
B W W

.....[370]......
B B B
_ ? B
B W W

.....[371]......
B B B
_ B ?
B W W

.....[372]......
? B B
_ B B
W W _

.....[373]......
B ? B
_ B B
W W _

.....[374]......
B B ?
_ B B
W W _

.....[375]......
B B B
? B B
W W _

.....[376]......
B B B
_ ? B
W W _

.....[377]......
B B B
_ B ?
W W _

.....[378]......
B B B
_ B B
? W _

.....[379]......
B B B
_ B B
W ? _

8.2. OUTPUT OF DIAMOND PATTERNS WITH K_BITS AND FREQUENCY

```
    _
  _ B W
W B B W B
  W B W
    W
FREQ    -->8
K_VALUE_ -->11

    B
  W B B
B W B _ _
  W W B
    _
FREQ    -->8
K_VALUE_ -->10

    B
  W B B
_ _ _ B W
  _ _ B
    B
FREQ    -->8
K_VALUE_ -->8

    W
  B W B
B W B B _
  B W B
    W
FREQ    -->19
K_VALUE_ -->12

    W
  W B W
W B B _ W
  W B B
    W
FREQ    -->18
K_VALUE_ -->12

    _
  W B B
W B W B W
  _ W W
    _
FREQ    -->11
K_VALUE_ -->10

    B
  _ B W
_ _ B B W
  B _ _
    B
FREQ    -->13
K_VALUE_ -->8

    B
  B W B
_ B B W B
  _ _ W
    _
FREQ    -->11
```

```
K_VALUE_ -->9

    B
  B B _
B W B B B
  W W W
    _
FREQ    -->44
K_VALUE_ -->11

    B
  B W B
_ _ W W B
  W _ _
    _
FREQ    -->12
K_VALUE_ -->8

    B
  W B W
_ W W B _
  _ _ W
    _
FREQ    -->10
K_VALUE_ -->8

    _
  B _ _
_ _ _ B _
  W W W
    B
FREQ    -->8
K_VALUE_ -->6

    B
  W W B
W _ _ W W
  _ _ _
    W
FREQ    -->23
K_VALUE_ -->8

    B
  _ _ _
W W W W W
  W B B
    B
FREQ    -->22
K_VALUE_ -->10


    _
  _ _ B
W W W W B
  B B B
    _
FREQ    -->32
K_VALUE_ -->9

    _
  _ B _
W W W B _
  B B _
```

```
    _
FREQ    -->10
K_VALUE_ -->7

    B
  B _ _
W W B _ B
  B _ _
    _
FREQ    -->23
K_VALUE_ -->7

    _
  _ _ _
_ _ _ W W
  W W B
    W
FREQ    -->64
K_VALUE_ -->6

    _
  _ _ _
_ _ W W _
  W B B
    B
FREQ    -->37
K_VALUE_ -->6

    _
  _ _ _
_ W W _ W
  B B W
    W
FREQ    -->11
K_VALUE_ -->7

    _
  _ _ _
W W _ W W
  B W _
    W
FREQ    -->10
K_VALUE_ -->7

    _
  _ _ W
W _ W W B
  W _ B
    B
FREQ    -->10
K_VALUE_ -->8

    W
  _ W B
_ W W B _
  _ B W
    B
FREQ    -->12
K_VALUE_ -->9

    W
  W W _
W B B B B
```

```
  W B W
    W
FREQ    -->17
K_VALUE_ -->12

    W
  _ B W
B B B W _
  W B B
    B
FREQ    -->9
K_VALUE_ -->11

    _
  _ _ W
_ W W B B
  B W B
    B
FREQ    -->16
K_VALUE_ -->9

    _
  _ W W
W W B B W
  W B W
    B
FREQ    -->20
K_VALUE_ -->11

    _
  W W _
W B B W _
  B W W
    W
FREQ    -->15
K_VALUE_ -->10

    _
  W _ W
B B W _ B
  W W B
    W
FREQ    -->10
K_VALUE_ -->10

    B
  _ W B
B B B W B
  B W W
    W
FREQ    -->10
K_VALUE_ -->12

    W
  W B B
B B W B W
  W W W
    _
FREQ    -->75
K_VALUE_ -->12

    W
  B B B
```

```
B W B W B
  W W B
    _
FREQ    -->14
K_VALUE_ -->12

    _
  B B B
W B W B B
  W B B
    W
FREQ    -->12
K_VALUE_ -->12

    B
  B B W
B W B B W
  B B B
    W
FREQ    -->19
K_VALUE_ -->13

    _
  W W B
W B W B W
  W B B
    B
FREQ    -->14
K_VALUE_ -->12

    W
  W B B
B W B W W
  B B W
    _
FREQ    -->16
K_VALUE_ -->12

    W
  B B W
W B W W B
  B W W
    W
FREQ    -->18
K_VALUE_ -->13

    _
  B B B
_ _ B W W
  _ B W
    B
FREQ    -->11
K_VALUE_ -->9

    W
  B B W
_ B W W W
  B W _
    B
FREQ    -->34
K_VALUE_ -->11

    B
```

```
B W B
B W W W B
W _ _
W
FREQ     -->21
K_VALUE_ -->11

B
W B W
W W W B B
_ _ W
_
FREQ     -->31
K_VALUE_ -->10

B
B W B
W W B B B
_ W W
W
FREQ     -->18
K_VALUE_ -->12

B
W B B
W B B B B
W W W
_
FREQ     -->17
K_VALUE_ -->12

W
B W B
W W B B W
B B _
W
FREQ     -->17
K_VALUE_ -->12

B
W B W
W B B W W
B _ W
W
FREQ     -->10
K_VALUE_ -->12

B
B B _
W W W B _
B W B
B
FREQ     -->20
K_VALUE_ -->11

B
_ _ B
W B _ B W
B _ B
B
FREQ     -->12
K_VALUE_ -->9

B
_ B W
B _ B W _
_ B B
W
FREQ     -->8
K_VALUE_ -->9

B
B W W
_ B W _ _
B B W
W
FREQ     -->39
K_VALUE_ -->10

W
W B B
W B B _ W
B W W
B
FREQ     -->33
K_VALUE_ -->12

B
_ B W
B _ B B W
B W W
_
FREQ     -->19
K_VALUE_ -->10

W
B W _
_ B B W _
W W _
W
FREQ     -->11
K_VALUE_ -->9

_
B _ B
B _ B W W
_ B _
_
FREQ     -->8
K_VALUE_ -->7

_
B W _
W W _ _ _
W W W
B
FREQ     -->13
K_VALUE_ -->8

B
B W W
_ _ B W B
B B W
B
FREQ     -->12
K_VALUE_ -->11

B
B W W
_ B _ W W
_ B W
B
FREQ     -->9
K_VALUE_ -->10

B
W W _
B _ W W W
B W B
B
FREQ     -->12
K_VALUE_ -->11

W
_ _ W
W W W W W
B B _
B
FREQ     -->9
K_VALUE_ -->10

W
_ B W
W B B W _
_ B W
B
FREQ     -->11
K_VALUE_ -->10

W
B W B
B B W _ W
B W W
B
FREQ     -->11
K_VALUE_ -->12

W
W B B
B W _ W W
W W W
B
FREQ     -->16
K_VALUE_ -->12

_
B B B
W _ W W B
W W B
W
FREQ     -->10
K_VALUE_ -->11

B
B B W
_ W W B _
W B B
W
FREQ     -->14
K_VALUE_ -->11

W
B _ W
_ _ B W B
B B B
W
FREQ     -->11
K_VALUE_ -->10

B
B B W
_ _ B W W
_ B B
B
FREQ     -->21
K_VALUE_ -->10

W
B W _
_ B W W W
B B B
W
FREQ     -->16
K_VALUE_ -->11

B
W _ W
B W W W B
B B W
W
FREQ     -->35
K_VALUE_ -->12

B
_ W W
W W W B B
B W W
B
FREQ     -->32
K_VALUE_ -->12

B
W W B
W W B B B
W W B
W
FREQ     -->69
K_VALUE_ -->13

W
W B _
W B B B B
W B W
W
FREQ     -->68
K_VALUE_ -->12

_
B _ _
B B B B B
B W W
_
FREQ     -->22
K_VALUE_ -->9

B
_ _ B
B B B B B
W W W
W
FREQ     -->24
K_VALUE_ -->11

B
_ B W
_ _ B B B
_ B W
B
FREQ     -->28
K_VALUE_ -->9

W
B W W
_ B B B W
B W W
_
FREQ     -->19
K_VALUE_ -->11

_
W W W
B B B W W
W W B
W
FREQ     -->10
K_VALUE_ -->12

W
W W B
B B W W B
W B W
B
FREQ     -->48
K_VALUE_ -->13

W
W B B
B W W B W
B W W
W
FREQ     -->35
K_VALUE_ -->13

B
B B B
W W B W W
W W _
_
FREQ     -->30
K_VALUE_ -->11

_
B B B
B W W W B
_ W B
_
FREQ     -->15
K_VALUE_ -->10

B
B B B
W W W B W
W B B
W
FREQ     -->45
```

```
K_VALUE_ -->13

B
B _ _
W B W _ _
B B B
W
FREQ     -->8
K_VALUE_ -->9

W
B B W
B W W B W
_ W B
W
FREQ     -->20
K_VALUE_ -->12

W
B W W
W W B W W
W B W
B
FREQ     -->27
K_VALUE_ -->13

B
W W B
W B W W _
B W _
W
FREQ     -->19
K_VALUE_ -->11

B
W B W
B W W _ W
W _ _
_
FREQ     -->33
K_VALUE_ -->9

B
B W W
W W _ W B
_ _ _
W
FREQ     -->10
K_VALUE_ -->9

B
W W W
W _ W B B
_ _ W
W
FREQ     -->21
K_VALUE_ -->10

B
W B W
W B B B B
W B W
B
FREQ     -->21
K_VALUE_ -->13

_
W _ _
B B B B B
W W W
_
FREQ     -->12
K_VALUE_ -->9

B
B W W
B B _ W B
W W W
_
FREQ     -->11
K_VALUE_ -->11

B
W W B
B _ W B W
W W B
W
FREQ     -->10
K_VALUE_ -->12

W
W B W
_ W B W _
W B W
B
FREQ     -->14
K_VALUE_ -->11

B
W W W
B W _ _ W
B B B
_
FREQ     -->14
K_VALUE_ -->10

W
W W B
W _ W B B
W B B
B
FREQ     -->12
K_VALUE_ -->12

_
W _ W
B B W B W
W W B
B
FREQ     -->13
K_VALUE_ -->11

_
_ W W
B W B W B
W B B
B
FREQ     -->10
K_VALUE_ -->11

W
W W B
B W B _ B
B B _
_
FREQ     -->26
K_VALUE_ -->10

W
B W _
B _ B B B
_ _ _
_
FREQ     -->14
K_VALUE_ -->7

_
_ _ _
_ _ W _ B
_ W B
W
FREQ     -->37
K_VALUE_ -->5

_
_ _ _
_ W _ B _
W B _
_
FREQ     -->19
K_VALUE_ -->4

_
_ _ W
_ B _ _ W
_ W _
_
FREQ     -->10
K_VALUE_ -->4

_
_ W _
B _ _ W _
W _ _
_
FREQ     -->14
K_VALUE_ -->4

_
W _ _
_ _ W _ _
_ _ B
W
FREQ     -->13
K_VALUE_ -->4

_
_ _ _
W _ _ W _
B _ _
B
FREQ     -->10
K_VALUE_ -->4

_
_ _ W
_ _ W _ B
_ _ B
_
FREQ     -->14
K_VALUE_ -->4

_
W _ _
W _ B _ _
B _ _
_
FREQ     -->8
K_VALUE_ -->4

_
W _ B
_ W B _ _
W _ _
_
FREQ     -->9
K_VALUE_ -->5

_
_ B _
W B _ _ _
_ _ _
B
FREQ     -->34
K_VALUE_ -->4

_
_ _ W
_ B _ _ B
_ B _
_
FREQ     -->26
K_VALUE_ -->4

_
_ W _
B _ _ B _
B _ _
_
FREQ     -->18
K_VALUE_ -->4

B
B _ _
W _ _ _ _
_ B _
_
FREQ     -->33
K_VALUE_ -->4

_
_ B _
B _ _ _ W
_ _ B
_
FREQ     -->13
K_VALUE_ -->4

B
B _ _
_ _ _ W _
_ B W
B
FREQ     -->15
K_VALUE_ -->6

B
_ _ _
_ _ _ B W
_ _ B
_
FREQ     -->167
K_VALUE_ -->4

_
_ _ W
_ _ B W _
_ B W
B
FREQ     -->42
K_VALUE_ -->6

_
_ B W
_ _ B W _
_ B W
_
FREQ     -->24
K_VALUE_ -->6

_
_ _ B
B _ _ B W
_ _ _
W
FREQ     -->10
K_VALUE_ -->5

_
_ _ _
_ _ _ _ B
_ B B
W
FREQ     -->145
K_VALUE_ -->4
```