CS4444: Project 3 - Cluedo - Group 1 Final Report

Mark Ayzenshtat, Hanhua Feng, and Stephen Lee

November 6, 2003

1 Overview

Cluedo, the deductive game with n cards, p players, c cards per player, and k hidden cards, is in many ways similiar to the Rectangles game. Unlike the Cookie Cutter problem, Rectangles and Cluedo involve direct interaction between players. In both games, players may try to destroy the constructive efforts of other players. Moreover, some of the strategies suggested during class discussions about Cluedo seemed reminiscent of those discussed and implemented for Rectangles. However, unlike Rectangles, Cluedo players do not have to divide their time and energy into being destructive. Cluedo players have plenty of opportunities to be as constructive or destructive as they choose. The only hard limit is the time spent computing the values involved with each of their respective strategies.

A few classes before the tournament, it became apparent that many players had matured to the point of exhaustively inferring almost all the information that could be inferred from all player actions during a game. These players had found all absolute truths. Additionally, it seemed as if the ranking difference between players was often simply determined by the move order. Clearly, success in the tournament rankings would be achieved by the players who could make correct guesses before all the other players had enough information to confidently guess.

Our players could gain an advantage over other players if we made some assumptions about our opponents. Group1Player2 puts its assumption into use early in the game. Group1Player3 tries to guess early based on the assumption that a player, P_i will probably not ask another player, P_j about cards which P_i already knows P_j owns.

2 Preliminary Work

Our first attempts at designing a successful Cluedo player incorporated a probability matrix. One dimension of the matrix would represent all players (including the "player" for the hidden cards); the other would represent all cards. Each cell could take on a real number value between 0.0 and 1.0 that would indicate, at any point during a game, the likelihood that a given player possesss a given card. Various player actions (e.g. interrogations, responses, guesses) would modify these probabilities when they occurred. Presumably, our player would then use the values in this table to issue interrogations with the highest possible chance of yielding new information and guesses with the highest possible chance of being correct.

Before we could implement this kind of decision scheme, we needed to figure out precisely how player actions over the course of a game would be "remembered" and how they would specifically affect the probability matrix. In class, we observed that it was possible to represent the constraints on player-card ownership as a set of logic expressions. We decided to maintain such an expression set and with it encapsulate all of the knowledge our player has about other players' card possessions. Then, each round, we would assign probabilities to the matrix based on these clauses, update the matrix as needed, and then formulate decisions based on the resulting probabilities.

To realize this multi-stage decision scheme, we initially made use of the Orbital logic library. Our first player defined a logic term for each player/card combination (e.g. the term representing whether player 3 possessed card 4 would be $p3c4$). For each action, the player would create a logic expression comprised of such terms. For example, after player 2 makes a guess, all of its cards are revealed. When this occurs, our player would create two expressions: one to signify that player 2 possesses all of the revealed cards:

$$(p2c3\&p2c5\&p2c7\&p2c9) \tag{1}$$

and the other to signify that all other players do not possess those same cards:

$$!(p1c3\|p3c3\|p4c3\|p5c3\|...) \tag{2}$$

The player would maintain a master list of these expressions as premises. Resolving these premises, it would then assign each player/card term a value of *true*, *false*, or *unknown*. From these discrete assignments, we suspected that it would then be possible to generate the real number probabilities that would populate the final matrix.

We soon realized, however, that it was quite difficult to correctly "balance" the matrix. Whenever a single cell value was changed, it required numerous other values to be updated as well in order to preserve row and column sums. Effectively, every cell update potentially necessitated a simultaneous reevaluation of the entire matrix. We ultimately agreed that it would be easier to use a decision table of discrete values (and possibly lose some decision-making capability) than to devote too much time trying to figure out how to efficiently carry out this reevaluation.

3 Methods

We submitted two players to the tournament. The first player, "Super Crazy Broomhead" used a totally deterministic logic model, while the second player "Psychotic Broomhead" used a probabilistic one.

3.1 "Crazy Broomhead": A Conservative, Baseline Player

This player contains two parts: Counting Logic and the Interrogation Analyzer. The two parts are somewhat independent; the Counting Logic totally ignores the existence of the interrogation strategies, while the Interrogation Analyzer more or less treats the Counting Logic as a black box which provides some state variables.

3.1.1 Counting Logic

Although we originally intended to use the Orbital library mentioned above for this purpose, we soon realized that the Counting Logic could be implemented without using the vast number of options provided by Orbital. Complete logic inferences using Orbital consume a lot of memory and are slow to compute. Thus, we implemented our own logic class, **CountingLogic**.

3.1.2 Data Structures

The Counting Logic class contains a matrix and a list. One dimension of the matrix spans the cards, and the other enumerates the players. The values represent an ownership state, which details whether the card is owned, unowned, or carries an unknown status with respect to the player. Each entry of the matrix has three possible values: positive, negative, and zero. A positive value indicates that the player holds this card. A negative one indicates that the player does not. Lastly and most importantly, a zero value means the relationship between this player and this card is uncertain.

In addition to this matrix, the Counting Logic class maintains two arrays: one indicates how many cards each player holds and the other indicates how many players each card can be held by. The latter array is filled by ones. We made the latter array more general such that a card can be held by more than one player, standard playing cards has four copies of each card, which might make this problem more interesting.

The list contains unresolved clues. A clue contains four elements: a list of cards, a player index, a lower bound and an upper bound of the number indicating how many cards this player is possibly holding from the list of cards. Again, we made this problem more general here.

3.1.3 Clue Reduction

A clue can be reduced by removing known facts from the list according to the matrix. If the corresponding entry of a removed card in the matrix is positive, the lower bound and the upper bound were decremented by one. After reduction, the clue would be in one of four states:

- The clue is useless if the lower bound is less than or equal to zero, and the upper bound is greater than or equal to the length of the list. In this case, this clue is discarded.
- The clue is deterministic if the lower bound is greater than or equal to the length of list, or, the upper bound is less than or equal to zero. In the first scenario, this player holds all this cards, and in the second, this player holds none. Then the corresponding matrix elements are updated and this clue is discarded.
- The clue is illegal if the lower bound is greater than the upper bound. In this case, incongruity of the logic database is found, the current implementation throws a **InvalidClueException**.
- Otherwise, the clue is unresolved. This clue will be stored in the list of unresolved clues for future uses. Meanwhile, the program will do two additional examinations: first checking the lower bound of the clue against the number of certainly held cards of this player. It adds these two numbers. If the sum is equal to c, then cards in the list cannot possibly be held by this player, and the program can update the matrix accordingly. (If the sum is greater, that means a incongruity of the database, an exception will be thrown). Secondly the program checks the upper bound of the clue against the number of certainly unheld cards, and a similar operation is performed.

3.1.4 Interrogation Analyzer: Clue Addition to Counting Logic

When a clue is added to the Counting Logic class, we do the following:

- Reduce the clue according to the matrix.
- If this clue is still unresolved, add this clue into the unresolved list.
- If the matrix is changed, propagate this changed horizontally and vertically according to the horizontal and vertical constraints, until no further changes can be made.
- If the matrix is changed, try to reduce all unresolved clues. If the matrix is changed again during reduction, repeat the previous step.

3.1.5 Additional Inferences

Although the processing behind the Counting Logic is simple and efficient, the list of inferences that can be made is not complete. Therefore, we add one more reduction method. Crazy Broomhead will arbitrarily choose a card from the shortest unsolved clues and try to set the corresponding entries of the matrix to both positive and negative, and then propagate this change in the matrix to the list of unsolved clues. After

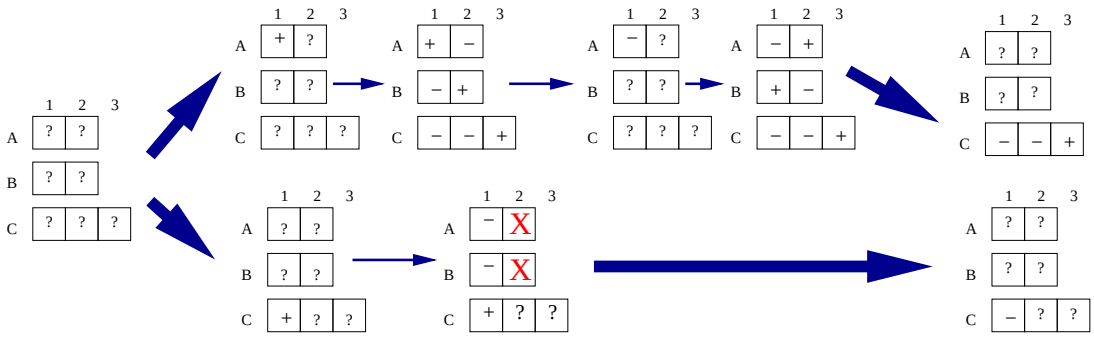

Figure 1: Illustration of our additional inference algorithm. Suppose player A has one of 1 and 2, B has one of 1 and 2, and C has one of 1, 2 and 3. Initially we do not know who has which card, as indicated in the leftmost figure. Then, by following the upper path, we try to let player A hold 1 or not hold 1. In both cases, we find, after propogation, that player C always has 3 but not 1 and 2. Therefore, three more entries are determined for player C. Alternatively, we can follow the lower path and find that if we let player C hold 1, an inconguity would be found during propogation. Therefore, player C should not hold 1. A "+" means we are certain that the corresponding player holds the corresponding card, a "-" means we are certain the it does not, and a "?" means we are not certain. The red "X" means a inconguity was found.

propagation, if some entries are originally zero and repeatedly resolves to the same non-zero value for every trial, then this entry is deterministic. Crazy Broomhead will change the original matrix accordingly. If an "InvalidClueException" is caught during trial, the program will set this entry to the alternate value. Figure 1 illustrates how this algorithm works.

3.1.6 Undeniable and Probable Clues

We classified clues into two types, undeniable clues and probable clues. The fol lowing are undeniable clues:

- Cards held by our own player (lower bound = upper bound = c).
- A player responded negatively to an interrogation (lower bound = upper bound = 0).
- A player responded affirmatively to our player's interrogation (number of cards = 1, lower bound = upper bound = 1).
- A player exits the game (number of cards = c, lower bound = upper bound = c).
- A player responded affirmatively to another player's interrogation (lower bound = 1, upper bound = number of cards in the list).

All clues from other sources are probable clues.

3.1.7 "Offense": Interrogation Strategies

Before interrogation, we decide whom to interrogate. Super Crazy Broomhead chooses the player holding the most number of uncertain cards as its next target of interrogation. However, Crazy Broomhead will arbitrarily designate a player to be unmolested for the purpose of the end-game strategy. Once a target has been selected, Super Crazy Broomhead has two strategies of interrogation: the open-game strategy and the end-game strategy. If there is only one player holding uncertain cards, the program uses end-game strategy. Therefore, in two-player games, the program only employs the end-game strategy.

- **Open-game Strategy**: Crazy Broomhead asks for all uncertain cards of this player. The interrogatee must response positively.
- **End-game strategy**: Crazy Broomhead asks for a subset of k uncertain cards. If it gets a positive answer, it continues with the same open-game strategy. If it receives a negative reply, it prepares to guess next round, since it has enough information about all the hidden cards.

3.1.8 "Defense": The Padding Strategy

All cards besides those certainly held by the interrogatee might be added to the padding list. For the open-game strategy, the chance that a card is added to the padding list is three out of four; for the end-game strategy, it is one out of four, since we do not want other players to get more information from our interrogation.

3.1.9 "Defense": Response Strategy

Ideally, the Crazy Broomhead player should hide as much information as possible. Thus, Crazy Broomhead prefers to respond to an interrogation with a card that has been disclosed. Otherwise, if it must disclose a new card, it should respond with a random card. Internally, it maintains two lists of its own cards. One is the disclosed, initially empty list. The other contains the remaining cards, an initially randomized list of its own cards. If Crazy Broomhead becomes the subject of an interrogation, it will first try to respond with a disclosed card. If it cannot find one, it will use the first card in the undisclosed list and move this card to the tail of the disclosed list.

3.1.10 Guessing Condition

If the rate of success is greater than 95%, the Crazy Broomhead makes a guess according to the matrix. In other words, if the number of possible combinations of k hidden cards is less than 20 out of 19, Crazy Broomhead guesses. All cards whose corresponding value are positive will be added to the guess list. A guess list is made up by the uncertain cards for the remaining guess slots. For the non-probabilistic implementation, 95% actually means 100% certain.

3.2 Group1Player2: "Super Crazy Broomhead"

The original players Broomhead and Crazy Broomhead use only undeniable clues. However, we found every player seems to be computing the logic well. Inevitably, game results would be quite random. Consequently, we implemented "Super Crazy Broomhead". It assumes most of the players would interrogate other players using a long list and wait for a positive response in multiple player game. During the first round, when players are only aware of their own cards, they typically ask for all unknown cards and pad the list with some of their own cards. Therefore, we assume that all not present in its first interrogation are held by the the interrogator. Super Crazy Broomhead utilizes this probable clue only for the first round; afterward, it resumes its usual interrogation strategy. Of course, no player can hold more than c cards. If, during the first round, a player interrogates another player about a set with fewer than $n - c$ cards, Super Crazy Broomhead ignores this interrogation. Thus, Super Crazy Broomhead collects a lot of information during the first round, much more than most other players do. Meanwhile, the original Clue Logic class runs simultanenously. Once Super Crazy Broomhead finds anf incongruity in the counting logic, it will return to the original Clue Logic used by Crazy Broomhead. This might be the only reason that Group1Player2 dominates all games that have more than two players.

Super Crazy Broomhead is not used by Group1Player2 for two-player games. In these two-player games, Group1Player2 did relatively well. We probably should be grateful for our end-game interrogation strategy.

Number of Players	1 C/P		2 C/P			3 C/P			5 Cards/Player				8 C/P				16 C/P			
	Number of Hidden Cards																			
	1	**2**	**1**	**2**	**4**	**1**	**3**	**6**	**1**	**2**	**5**	**10**	**1**	**4**	**8**	**16**	**1**	**8**	**16**	**32**
2	4	5	2	1	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	2	1	1	1	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	2	1	1	1	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	3	1	1	1	1	2	1	1	1	1	1	1				
9	1	1	1	1	2	1	2	3	1	2	2	1	1	1	1	1				
10											1									

Table 1: Average Rankings of Super Crazy Broomhead with respect to Game Parameters

3.3 Group1Player3: "Psychotic Broomhead"

Like Super Crazy Broomhead, Psychotic Broomhead is a branch from Crazy Broomhead. Psychotic Broomhead made the fairly conservative assumption that once another player, P_i, has interrogated yet another player, P_j, and finds that P_j owns card, C_a, it is unlikely that P_i will ask P_j about C_a again. If P_i asks P_j about C_a again, P_j should probably return C_a. Ultimately, P_i basically will gain nothing from a second interrogation which includes C_a. If the previous assumption holds true, with enough interrogations conducted by every player combination and the absolute information we derived from our own interrogations and the false interrogations, we should have a decent idea of which cards are owned by which players.

Psychotic Broomhead maintains a p by n table, T_i, for each player, P_i. Each time P_i takes a turn, Psychotic Broomhead ages each element in T_i except the cells which represent the cards that P_i interrogated P_j about. For example, assuming Psychotic Broomhead is P_0, if P_1 asks P_3 about cards, $0, 3, 5$ and P_3 acknowledges having a card, C_a, from that set, then every cell of T_1 would be incremented except cells, $(3, 0)$, $(3, 3)$, and $(3, 5)$, which are reset to zero. Theoretically, P_1 might interrogate P_3 again and inevitably throw "garbage" into its interrogation to throw off the other observing players. Since our assumption states that P_1 should not include C_a, C_a will definitely age while all the other cards will potentially be reset.

To further supplement this interpretation of other players' positive interrogations, Psychotic Broomhead guesses the c cards which P_j owns by collecting the c oldest cards from row j of each T_i where $i! = j$ and $i > 0$. From this set of $c(p-2)$ cards, we select the most frequently appearing $(c - \pi)$, where π represents the cards which Psychotic Broomhead knows P_j definitely owns.

Psychotic Broomhead collects all the absolute truths that its counterpart, Crazy Broomhead, infers. Thus, if Psychotic Broomhead makes an early guess right before everyone else, it is only fairly uncertain about the set of cards owned by one or two opponents. Clearly, this approach relies heavily on a sufficient number of interrogations. This probably occurs in games with many players and many cards, with an inclination towards a low k.

4 Results

While Psychotic Broomhead's performance paled in comparison to Super Crazy Broomhead's, both players served as interesting offshoots from the same basic player, Crazy Broomhead.

4.1 Super Crazy Broomhead

Table 1 indicates that Super Crazy Broomhead's assumptions proved to be very fruitful. When Super Crazy Broomhead did not come in first, it usually fell right behind Psychotic Broomhead.

Perhaps, the more interesting results were the two player games with one or two cards per player. In

Number of Players	1 C/P		2 C/P			3 C/P			5 Cards/Player				8 C/P				16 C/P			
	Number of Hidden Cards																			
	1	**2**	**1**	**2**	**4**	**1**	**3**	**6**	**1**	**2**	**5**	**10**	**1**	**4**	**8**	**16**	**1**	**8**	**16**	**32**
2	9	9	9	9	9	7	8	7	6	6	8	6	5	6	7	7	6	7	7	7
3	9	9	6	4	6	2	3	7	7	6	6	6	7	8	7	7	8	8	8	7
5	9	8	4	5	5	3	2	2	2	2	2	7	9	8	8	8	9	8	8	8
7	9	8	5	3	4	5	4	3	2	1	2	4	9	4	4	7				
9	9	8	4	5	5	4	1	2	2	1	1	2	3	2	2	8				
10											5									

Table 2: Average Rankings of Psychotic Broomhead with respect to Game Parameters

those games, Super Crazy Broomhead performed average, which probably means other players were being more aggressive in guessing early or our Crazy Broomhead, our most conversative player whose sole purpose was to extract only the absolute truth, was not extracting enough information relative to the other players.

4.2 Psychotic Broomhead

Table 2 indicates that Psychotic Broomhead performed significantly better with more cards than with few cards. We suspect that Psychotic Broomhead threw Exceptions during games with greater than eight cards. For games with greater than eight cards per player, we clearly see its performance degrade from eight cards games to sixteen cards games. Otherwise, with the exception of games with one card per player, as the number of players per game increases, Psychotic Broomhead gained enough data to make some fairly accurate guesses as to which cards were owned by each player.

5 Conclusion

Excluding player bugs, Psychotic Broomhead's seemingly more conservative assumptions proved to be far less effective compared to the simpler, stronger assumptions made by Super Crazy Broomhead. Super Crazy Broomhead's end-game probably edged its' game past opponents, including Psychotic Broomhead.

Ultimately, the players presented by each group clearly showed that the entire class was close to exhausting the number of inferences that could be derived from each of their respective interrogations. To gain an edge, assumptions that were consistent across all the opponents had to be made. Generally, excluding cheating, in any closely competed match where players' actions can affect both the performances and countermoves of other players, a reasonable analysis of how the other contestants' play will be the only way to score the decisive points.

A Optimal number of interrogations in two-player clue games

In two-player games, the best solution for the worst permutation is to interrogate the opponent for c times, where c is the number of cards held by each player. This can be proved by adversary – just suppose the opponent can cheat by exchanging cards on hand with those on stake and always respond positively. Therefore, whatever interrogation strategy is used, there are always some permutations such that one has to do k interrogations. Similarly we can also prove $n - c - k$ is the best solution for the worst permutation in multi-player games, by adversary.

We now turn to study the optimal expected number of interrogations, since one cannot do better in the worst case. Only two-player games are discussed here for simplicity.

Suppose p cards held by the opponent and q cards on the stake are uncertain for our player. The goal of our player is to find the optimal expectation of number of interrogations, and the optimal number of cards it is going to ask in the next turn. Let $Q(p,q)$ be the optimal expected number of interrogations, and $Q(p,q,k)$ be the optimal expected number of interrogations on the condition that we are going to ask for k cards in the next turn. Clearly,

$$Q(p,q) = \min_{1 \le k \le p+q} Q(p,q,k) \tag{3}$$

and

$$Q(p,q,k) = \begin{cases} [1-\alpha(p,q,k)]Q(p-1,q) + \alpha(p,q,k)Q(p,q-k) + 1 & 1 \le k \le q \\ Q(p-1,q) + 1 & q < k \le p+q \end{cases}, \tag{4}$$

where

$$\alpha(p,q,k) = \frac{\binom{q}{k}}{\binom{p+q}{k}}. \tag{5}$$

Also, we have boundary conditions that $Q(0,q) = 0$ and $Q(p,0) = 0$.

More mathematics can be done here; but this problem is now a *static* dynamic programming problem, whose solution is independent to the permutation of the cards. So we wrote a program to solve the problem. Entries of the following table, generated by our program, indicate the expected numbers of interrogations and optimal numbers of cards for the next interrogation for different ps and qs.

p: #cards on hand	q: #hidden cards									
	1	2	3	4	5	6	7	8	9	10
1	1.0/1	1.0/2	1.0/3	1.0/4	1.0/5	1.0/6	1.0/7	1.0/8	1.0/9	1.0/10
2	1.667/1	1.833/2	1.9/3	1.933/4	1.952/5	1.964/6	1.972/7	1.978/8	1.982/9	1.985/10
3	2.25/1	2.65/2	2.805/3	2.878/4	2.918/5	2.941/6	2.956/7	2.966/8	2.973/9	2.978/10
4	2.8/1	3.473/2	3.725/3	3.837/4	3.894/5	3.927/6	3.947/7	3.96/8	3.969/9	3.975/10
5	3.333/1	4.308/2	4.658/3	4.807/4	4.879/5	4.918/6	4.942/7	4.957/8	4.967/9	4.974/10
6	3.857/1	5.154/2	5.592/2	5.775/3	5.863/4	5.91/5	5.937/6	5.954/7	5.965/8	5.973/9
7	4.375/1	5.981/1	6.51/2	6.741/3	6.848/4	6.903/5	6.934/6	6.952/7	6.964/8	6.972/9
8	4.889/1	6.763/1	7.422/2	7.707/3	7.835/4	7.897/5	7.931/6	7.951/7	7.963/8	7.972/9
9	5.4/1	7.515/1	8.33/2	8.675/3	8.823/4	8.892/5	8.929/6	8.95/7	8.963/8	8.971/9
10	5.909/1	8.247/1	9.237/2	9.645/3	9.81/3	9.887/4	9.926/5	9.949/6	9.962/7	9.971/8

As we can see in the table, the optimal number of cards for the next interrogation is less than or equal to q. Our submitted players would ask for exactly q cards for two-player games, which is almost optimal, except for combinations of small ps and large qs. An extreme case in this table is that one is better to ask for one card instead of two, if there are only two cards on the stake and more than six cards on the opponent's hand.

Finally, here is our program for this problem:

```
class OptQuest {
    static double binomial( int n, int c ) {
        if ( c > n - c )
            c = n - c;

        double ret = 1;
        for ( int i=0; i<c; i++ )
        {
            ret *= n - i;
            ret /= i + 1;
        }

        return ret;
    }
```

```
static double alpha( int p, int q, int k ) {
    return binomial( q, k ) / binomial( p+q, k );
}

static int solve( double[][] mat, int p, int q ) {
    int k = 0;
    double optx = 1E10;

    for ( int i=1; i<=p+q; i++ )
    {
        double x;
        if ( i <= q )
        {
            double alpha = alpha( p, q, i );
            x = (1-alpha) * mat[p-1][q] + alpha * mat[p][q-i];
        }
        else
        {
            x = mat[p-1][q];
        }
        x++;

        if ( x < optx )
        {
            optx = x;
            k = i;
        }
    }

    mat[p][q] = optx;
    return k;
}

static void solve( double[][] mat, int[][] k ) {
    int np = mat.length;
    int nq = mat[0].length;

    // Java initializes all elements to zero
    for ( int p=1; p<np; p++ )
        for ( int q=1; q<nq; q++ )
            k[p][q] = solve( mat, p, q );
}

public static void main( String[] args ) {
    int np = 11, nq = 11;
    double[][] mat = new double[np][nq];
    int[][] k = new int[np][nq];
    solve( mat, k );


    java.text.NumberFormat nf = java.text.NumberFormat.getInstance();
    nf.setMinimumFractionDigits( 1 );
    nf.setMaximumFractionDigits( 3 );
    nf.setMinimumIntegerDigits( 1 );
```

```
        for ( int p=1; p<np; p++ )
        {
            System.out.print( String.valueOf(p) );
            for ( int q=1; q<np; q++ )
                System.out.print( " & " + nf.format(mat[p][q])
                                  + "/" +  k[p][q] );
            System.out.println( " \\\\" );
        }
    }
}
```