Implement List

Part1. The implementation of the RPS

The implementation of the SPS

Part2. The method for generating KBU

Transform the KBU into CNF format and get the DIMACS array

Put DIMACS array into AST to judge if the clause is satisfiable

Implementation of the logical agent by combining SPS, AST, RPS

There are 3917 words in total, and the working time for this assignment is over one week.

Literature Review

Minesweeper is one of the classical games of the Windows operating system written by Robert Donner and Curt Johnson. It is popular with people because of although the rules of the game are simple, players need logic and arithmetic reasoning to make every movement. The birth of the minesweeper can back to 1980s; it appeared in the system of the Windows 3 series [1]. In 2000, According to Richard Kaye, the minesweeper was a game with the rectangular grid that has some numbers or dangers. Players need to open the covered cell to check if the cell contains danger then get the clue number. Moreover, Kaye proofed that the successful strategy of minesweeper is an NP-complete problem [2]. There are two features for the NP-complete problem. They are computable by a non-deterministic Turing machine in polynomial time, and every other NP-complete problem can be reduced in deterministic polynomial time to the particular NP-complete problem in question [3]. Like other NP problems (the travelling salesman), it is difficult to find an efficient method to address it. However, any NP problems can be reduced to SAT [4].

In this report, there are different agents to solve minesweeper problem by using three different algorithms, which is RPS, SPS, ATS.

Part One

1. The PEAS model

The general minesweeper problem is about some covered cell in a rectangular grid, and there are some dangers in the map. Players have to open the covered cell to get some clues which are some numbers around the cell has been uncovered. Players can know how many dangers around the current cell, then make the next movement. In this task, apart from the danger, there is also some gold in the map which means players can obtain one live when they open the gold cell. Players will win the game when they mark all the dangers before they have enough life because open the danger cell will make players lose one life.

- Performance measure

  The life of the player, players can gain lives when they open the gold cell and lose lives when danger cells are open.

- Environment

  The danger and the gold in the map.

- Actuators

  Probing the cell in the map which is covered and satisfies the search conditions from the top-left, line by line

  Uncovering the cell which is probing

  Marking the cell which is a danger

• Sensors

The clue (the numbers of the danger) around the uncovered cell

2. Implementation of The Game Infrastructure

At first, the game has to load the map and count the number of danger. The game also needs to know which cell is covered or uncovered and which one is marked as a danger, so there are arrays to store this information. There are some methods to help game open the covered cell or mark the cell by agent decisions. The player will win when all dangers are marked or lose the game when the life equal to 0. Therefore, the game should have methods to count the number of danger and the number of the marked cell.

3. Implementation of The Agent and Strategy

There are two strategies in the part one which is the random probing strategy and the single point strategy. The following words will introduce the content of each strategy and the implementation.

Random probing strategy RPS

The random probing strategy is an inefficient method because the success rate is very low. Like its name shows, it will open one cell of the map randomly without thinking if the cell is a danger or not.

As for the implementation, generating the x coordinate and the y coordinate by using random function, and the x and y cannot over the length of the map. Then, there will be some judgments about the probing cell. If the probing cell is gold, the live will plus one; if the probing cell is a danger, the life will dangers one; if the probing cell is a clue (numbers), opened the cell and stored it into uncover array. Keep probing the cell when the live equals 0 or the number of danger equals the number of the covered cell.

Single point strategy SPS

The single point strategy will open the covered cell or mark the cell depend on the clue around the probing cell. There are two situations that the single point strategy will do action. One situation is called all free neighbours (AFN) which means the clue of the cell is equals to the number of marked cells in its neighbours. In AFN case, the covered neighbours can be opened safely because all dangers around this cell have been marked. Another situation is all marked neighbours (AMN), which means the covered cell in the neighbour of the cell can be marked as danger when the number of the covered cell in the neighbour equals to the clue of the cell minus the number of the marked cell in neighbours. The SPS will mark cells of open cells only if there is a certain satisfaction in these two situations. To complete the game, the RPS will continue to probe cells when SPS stops work.

In terms of the implementation of the SPS. The first thing is to get the neighbours of the probing cell and add them into an array list. The implementation way is finding the valid neighbours coordinates by the coordinates of the probing cell firstly. Before adding them into the list, the coordinates have to be judged if the coordinates are in the map because in some case like the cell is on the border of the map, the coordinates generated by loop may out of the bound. The element added to the list is an object of the cell because this object cell can provide the information needed in other functions. Besides the method for neighbours, the most important thing is the method that defines the AFN and the AMN. In the AFN function, it needs a param to record the number of the danger. Then, traversing the uncovered neighbours, if there is a danger neighbour, the param of the danger initialised at first will plus one. After testing all the neighbour, if the clue of the cell equals the param danger, then the method returns true, otherwise return false. The implementation of the AMN is similar to the implementation of the AFN. In the AMN case, there are two params to record the number of the marked cell and the number of the covered cell. It will obtain these two number after traversing uncovered neighbours. Then, comparing the number of the covered cell with the clue of the cell minus the number of the marked cell, if they are equals to each other, then the function returns true, or else returns false. Besides these functions, there is another function called play, which implements the SPS by using the functions defined before. At first, uncover the top-left cell, then enter the do loop. In the content of the do loop, traversing the map to get the cell if the cell is covered. Then find the neighbours of this cell. After that, testing if there is a certain neighbour satisfy the AFN or the AMN. In the case of AFN, the cell will uncover; in the case of AMN, the cell will be marked. Otherwise, it will not do any action and continue to traverse next cell. There will be a param used to record if there are still cells are satisfying the AFN or AMN, it will plus one every time the function of AFN or AMN was called. The RPS will start work when the whole map was traversed and the time of method call is 0.

4. Evaluation of the Agents

The agent of the RPS selects one cell randomly that is covered in the map, so the rate for winning the game is very low. Especially in the case that there are many dangers in the map. However, as for the SPS, it can increase the success rate because it will not open the cell when it cannot reason the cell is danger or not. The agent of SPS will mark or uncover the cell only when it can distinguish the cell is safe or dangerous. However, the minesweeper problem cannot address only by using SPS, because SPS can only work when there is AMN or AFN situation in the game. The agent of SPS still needs RPS to complete the game. The chart below shows the number of the cell that needs to use RPS after the SPS in each map.

Map	Easy	Medium	Hard
Map1	5 Cells	0 Cells	120 Cells
Map2	16 Cells	10 Cells	111 Cells
Map3	0 Cells	0 Cells	63 Cells
Map4	0 Cells	56 Cells	0 Cells
Map5	21 Cells	39 Cells	73 Cells
Map6	0 Cells	58 Cells	135 Cells
Map7	10 Cells	6 Cells	106 Cells
Map8	10 Cells	44 Cells	132 Cells
Map9	0 Cells	4 Cells	104 Cells
Map10	0 Cells	58 Cells	135 Cells

Part Two

1. Implementation of The Agent and Strategy

SAtisfiability Test Strategy is the strategy that can reason next movement based on the knowledge base. In the minesweeper game, there will be some clues around the cell when the cell opened. These clues can regard as the knowledge that used to judge if there are some cells are danger. Hence generally, more information knowledge base involves, more dangers can be found. However, there are not many things can do if the knowledge base does not have enough information. Therefore, at the beginning of the game, it is not an appropriate way to use ATS to solve the minesweeper problem because of the limited knowledge base. Also, there will some cells cannot distinguish by ATS when there is no relevant information about the probing cell although the knowledge base is quite big. Thus, the logical agent should combine the SPS, the AST and the RPS. For example, at first, there is only one cell (left-top cell) in the map is uncovered. In this time, the information of the knowledge is very limited. Therefore, using the SPS to open more cells is a more efficient way because, after that, more clues will show in the map. Namely, more information can be involved in the knowledge base. As long as there is no more AFN or AMN situation in the map, that means the agent cannot use SPS to probe any more. Then it will create the knowledge base based on the open cell in the map. The knowledge base will not involve the cell that is marked as a danger or all its neighbours have been uncovered because this kind of cell cannot provide useful information for probing covered cells. The knowledge base will get the information from the open cell that has neighbours have not opened. The agent will use the knowledge to judge if there are some dangers in the map. Then, the agent will use SPS again when the ATS stop working because the knowledge does not have enough information. It may have a point that agent can use neither SPS nor ATS when there is no AFN or AMN in the map, and the knowledge base cannot provide useful information as well. Thus, the last way to complete the game is RPS. The agent will select the cell randomly, which is covered and is not marked as a danger in the map.

The crucial thing for the logical agent is the knowledge base. As mentioned before, the knowledge base will not involve all open cell, so the agent has to find the valid cell that can put into the knowledge base. For this, the agent will traverse all the open cell in the map, and judge the cell if it is not marked and it has neighbours that have not opened. After that, the agent will put these valid cells that will be used to generate knowledge base into an array list. There are some situations when the agent generates the knowledge base.

a. The value of the cell is 0

   In this case, all the covered neighbours around the cell are not dangerous. So, the generation sentence will like this:

▲ KBU = ~D01&~D02&~D21&~D13

b. #covered cell <= value - #marked cell

   In this case, all the covered neighbours around the cell will be dangers. So, the generation sentence will like this:

▲ KBU = D01&D21&D13

c. #covered cell > value - #marked cell

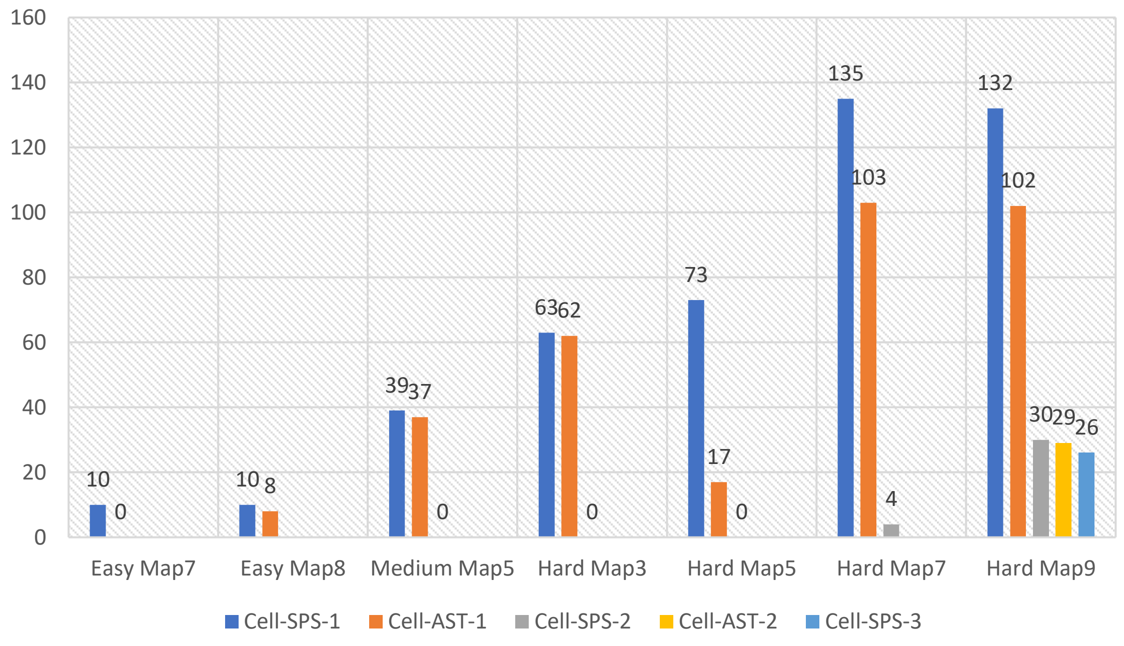
   This situation is a little complex because there are some different possibilities when the agent generates the knowledge base. The number of danger will depend on the value of the cell minus the number of the marked cell in its neighbours. For example, if the value of the cell is 3, and there has 1 danger been marked and still have 3 neighbours are covered. Therefore, the problem reduces to select 2 possible dangers in 3 neighbours and the number of the possibilities is 3 (3*2/2).

▲ KBU = D01&D02&~D21 | D01&D21&~D02 | D02&D21&~D01

The knowledge base constitutes by many characters, but the agent cannot understand the meaning of these sentences. Thus, two important tools are beneficial for the implementation of the logical agent. The key function of these tools is to help the agent understand the information of the knowledge base; then the agent will make the decision based on the reasoning results. The first tool is LogicNG; it can divide the knowledge base into the individual clause. In this procedure, the agent needs to delete the "D" in every clause and change the "~" into "-" by the replace function in object String. Then, the content of every clause will become the number, but the type is still String. To put the clause into an integer array, the clause has to transform the type advanced. The agent changes the knowledge base into the integer array by the LoginNG; then the SAT will reason the array. When the agent probes the cell, the SAT will judge if it is unsatisfiable for the knowledge base infer the cell is not a danger. Thus, the object solver of the SAT will put every clause into it and do the reasoning. If the problem is unsatisfiable, that means the probing cell is a danger; then the cell will be marked. The agent will start SPS again when AST cannot find more danger. Moreover, the agent will use RPS as long as both SPS and AST cannot do anything.

2. Evaluation of the Agents

The logical agent will find more danger cells in the map by AST based on the results of SPS. Therefore, the logical agent can reduce the number of cells that need random selection. The bar chart below shows the change of the covered cells' number by using various strategy. For example, in easy map 1, there are still 10 covered cells in the map after SPS, then the agent can find the danger cell by SAT and win the game. In the hard map 9, the agent calls the SPS three times and the AST two times, comparing to the SPS, the combination of two strategies greatly reduces the number of the random cell. Meanwhile, the rate of the winning can significantly enhance.



Examples and Test

To show the test result more clearly, the toString changed like the following code. Every step that is marking the cell or open the cell, the result will print out the specific cell. The "?" in the map represents the cell that has not opened. Moreover, the number is the clue of the cell, "X" represents the danger.

```
public String toString() {
	String s = "";
	for (int i = 0; i < this.N; i++) {
		for (int y = 0; y < this.N; y++) {
			if (this.mark[i][y]) {
				s += "X ";
			} else if(this.cover[i][y]) {
				s += "? ";
			} else {
				s += this.map[i][y] + " ";
			}
		}
		s += "\n";
	}
	return s;
}
```

```
reveal 3,6 life: 4
0 1 ? ? ? ? ? ?
0 1 X 2 1 1 0 0
0 1 1 3 X 3 1 0
0 0 g 3 ? ? 2 ?
0 0 0 3 ? ? ? ?
1 1 g 2 X 3 ? ?
X 2 1 1 1 2 1 1
? X 1 0 0 g 0 0
```

Random probing strategy RPS

The main function of the RPS is to generate the coordinate of the cell randomly. The implementation method is generating the random number by the function Math.Radom(). Due to the random number will between 0 to 1, which generated by the function. Therefor, it has to times the length of the map.

```
        x = (int) (Math.random() * game.map.length);

        y = (int) (Math.random() * game.map.length);
```

Single point strategy SPS

The key point for SPS is the implementation of two situations, which is AFN and AMN. Both of two situations need search the neighbours around the probing cell. The following code is the implementation of collecting the neighbours.

```
for (int x = m - 1; x <= m + 1; x++) {

        if (x >= 0 && x < game.map.length) {

                for (int y = n - 1; y <= n + 1; y++) {

                        if (y >= 0 && y < game.map[x].length) {

                                candinates.add(new Cell(x, y, game.map[x][y]));

                        }

                }

        }

}
```

As for the AFN, it needs to calculate the number of the marked cell in the neighbours by traverse the neighbours, then compare the number with the value of the cell. If two numbers are equal, the probing cell will trigger AFN. The agent will open the probing cell.

```
public boolean AFNRule(Cell c, Game game) {

        int dangerNum = 0;

        ArrayList<Cell> neighbours = getCandinates(c.x, c.y, game);

        for (int i = 0; i < neighbours.size(); i++) {

                if (game.mark[neighbours.get(i).x][neighbours.get(i).y]) {

                        dangerNum++;

                }

        }
        if (c.getValue() == dangerNum) {
              return true;
        } else {
              return false;
        }
}
```

In the term of AMN, the agent has to compare the number of the covered neighbours to the value minus the number of the marked neighbours. If the probing cell triggers the AMN, the agent will recognise it is danger and mark it.

```
public boolean AMNRule(Cell c, Game game) {
        int markNum = 0;
        int coverNum = 0;
        ArrayList<Cell> neighbours = new ArrayList<Cell>();
        neighbours = getCandinates(c.x, c.y, game);
        for (int i = 0; i < neighbours.size(); i++) {
              if (game.cover[neighbours.get(i).x][neighbours.get(i).y]) {
                    coverNum++;
              }
        }
        for (int i = 0; i < neighbours.size(); i++) {
              if (game.mark[neighbours.get(i).x][neighbours.get(i).y]) {
                    markNum++;
              }
        }
        if (c.getValue() - markNum == coverNum) {
              return true;
        } else {
                return false;
        }
}
```

SAtisfiability Test Strategy ATS

The AST relays on the knowledge base. Therefore, the vital function of the AST is the generation of the knowledge base. Also, to make the right reason, the agent has to read and understand the information of the knowledge base correctly. The following codes are the example of the generation format in different situations.

a. The value of the cell is 0

   ```
   KBU = "~D" + neighbours.get(i).x + neighbours.get(i).y;
   ```

b. #covered cell <= value - #marked cell

   ```
   KBU = "D" + neighbours.get(i).x + neighbours.get(i).y ;
   ```

c. #covered cell > value - #marked cell

   Due to every cell has different situation and value, the agent will use switch case to process situations in a different number of the possible dangers. The main method to generate knowledge base in diverse possibilities is using the loop to pick out one cell every time, and assume it is the danger. After that, the agent will generate a possibility by connecting all cells. For instance, if the number of danger in the neighbours is 3. The agent will calculate the possibilities initially by the combinatory formula:

   ```
   neighbours.size()*(neighbours.size()-1)*(neighbours.size()-2)/6;
   ```

   Then, the agent will pick out one cell:

   ```
   for(int i=0;i<neighbours.size();i++) {
           String subKBU_1="";
           subKBU_1 = "D" + neighbours.get(i).x + neighbours.get(i).y;
   ```

The agent will pick out the cell three times, after that, all three dangers have taken out of the neighbours, therefore others neighbours are all safe:

```
subKBU_4 = "&" + "~D" + neighbours.get(w).x + neighbours.get(w).y;
```

At last, all cells will connect to form a whole sentence:

```
possibility = "(" +subKBU_1 + subKBU_2 + subKBU_3 + subKBU_4 + ")";
```

After generating knowledge base, the agent will use LogicNG transform the content in the knowledge base to an integer array. To replace the simple, the agent should create an object String to store the clause. Then, the clause will be divided by "|", becoming to an array. Sometimes, there will be some String appear in the clause for some reason. The solution is replacing that element to a number; then the program can run normally. The integer array created by LogicNG will store in a list and transfer to SAT.

```
while(iterator.hasNext()){
        Formula clause=iterator.next();
String clauseStr = clause.toString();
        clauseStr = clauseStr.replaceAll("~", "-");
        String[] clas = clauseStr.split("\\|");
        dimacs = new int[clas.length];
        for(int i=0;i<clas.length;i++) {
                String trim = clas[i].trim();
                String trouble = "@RESERVED_CNF_";
                if(trim.contains(trouble)) {
                trim = "999999";
                }
        dimacs[i] = Integer.parseInt(trim);
        }
clauses.add(dimacs);
```

The SAT receives the list and adds every element in list into solver by traversing the list.

```
for (int i = 0; i < clauses.size(); i++) {
int[] claus = clauses.get(i);
	try {
		solver.addClause(new VecInt(claus));
	} catch (ContradictionException e) {
		e.printStackTrace();
	}
}
```

Then, the object problem will judge if the knowledge base can infer the probing cell is not a danger. If the problem is unsatisfiable, that means the probing cell is a danger; then the agent marks the probing cell.

```
IProblem problem = solver;
try {
	if (problem.isSatisfiable()) {
	} else {
		call++;
		int num = clauses.get(clauses.size() - 1)[0];
		String str = String.valueOf(num);
		String[] s = str.split("");
		int xcoordinate = Integer.parseInt(s[1]);
		int ycoordinate = Integer.parseInt(s[2]);
		game.mark(xcoordinate, ycoordinate);
	}
} catch (TimeoutException e) {
	e.printStackTrace();
}
```

Evaluation

For the implementation of part one, the order of the execution sequence is SPS first; then the RPS will start work when the SPS cannot do anything. However, the better way of the execution should recall the SPS when the RPS open a cell. If SPS cannot open more uncovered cells by the information of the cell, then continue to execute RPS. By this way, the agent can obtain more information from the cell. The same as part one, the implementation of part 2 also has this disadvantage. The agent will recall the SPS when the AST reviews more cell, however, it will not recall the first two strategy when it executes the RPS. Besides, there are too many codes for the implementation of the knowledge base; the code is not clear.

Running

The program needs a param when the user run it. The param is the name of the map in world class. And users can run the program by the command line: java -jar Logic1.jar [map]. There are three jar files in total, the Logic1 is the agent of the RPS, the Logic2 is the agent using SPS and RPS, the Logic3 is the agent combing SPS, RPS and SAT. For example, if user wants to test the SPS in medium map 3, the command line will be like this: java -jar Logic2.jar [MEDIUM3].

Bibliography

1. Studholme C. Minesweeper as a constraint satisfaction problem. Unpublished project report. 2000.
2. Pedersen K. The complexity of Minesweeper and strategies for game playing. Project report, univ. Warwick. 2004.
3. Castillo LP, Wrobel S. Learning minesweeper with multirelational learning. InIJCAI 2003 Aug 9 (pp. 533-540).
4. Kaye R. Minesweeper is NP-complete. The Mathematical Intelligencer. 2000 Mar 1;22(2):9-15.