CS5011 A02 Report

matriculation number: 160021429

1. Overview

Basically, the aim of this coursework is to implement logical agents with several strategies, which give additional information about the game to the logical agents. By using additional information, the logical agents decide what they will do next.

The game that we used for this project is called “Tornado Sweeper”. Tornado sweeper world is a grid in the shape of a rhombus with N cells per side, where each cell is of hexagonal shape.

The logical agent will win the game if and only if the logical agent probes all cells except T cells, where T is the number of tornado. If the logical agent probes the cell that has tornado in it, then the game will be over. And if the logical agent ends the game with more than T flagged cells, than the logical agent will lose the game. Also, the logical agent will lose the game if the logical agent flagged more than T cells.

1 - 1. Check Table

	Implemented
Part 1 - RPX	yes (RPX.java)
Part 2 - SPX	yes (SPX.java)
Part 3 - SATX	yes (SATX.java)
Extension1 - Add a new rule	yes
Extension2 - Triangular board	yes

1 - 2. Instruction (How to compile and run the program)

To compile my program, you first need to include the external jar files (LogicNG, antlr and sat4j) in your building path.

“export CLASSPATH=.:antlr-runtime-4.7.1.jar”
“export CLASSPATH=.:sat4j-pb.jar”
“export CLASSPATH=.:logicng-1.6.1.jar”

Then, you need to compile the java files to the class files by using the command below.

“javac -cp * *.java”

By doing this, you could compile the java files with the external jar files. Now, you could run my program.

“java -cp .:* A2main <RPX|SPX|SATX> <ID> <numberOfLifes>”

The command above is the basic usage of hte program. The first command line argument should be one of RPX, SPX and SATX. And the second command line argument should be the proper board name. The third command line argument is an optional one, but it should be a positive integer. Otherwise, you will get the error message.

Example commands:

1) java -cp .:* A2main SPX S2
2) java -cp .:* A2main RPX TEST1
3) java -cp .:* A2main SATX M3
4) java -cp .:* A2main SPX L2 5

2. Design and Implementation

2 - 1. Part 1 - RPX

The RPX is the logical agent that uses the random method to choose the cell to probe. To implement this, I simply used the Math.random() method to choose the random node on the world. The makeMove() method of the RPX class will check if the given coordinate is inspected, and probe that node if it is not inspected yet. If the probed node contains the tornado, then the program will print out suitable message, and it will be terminated.

2 - 2. Part 2

2 - 2 - 1. SPX

The SPX is the logical agent that uses the “Single point strategy”. Basically, the SPX will get the list of neighbours of the probed node, and check if one of the neighbours is either AFN (All Free Neighbours) or AMN (All Marked Neighbours). If the neighbour node is AFN, then the SPX will uncover the current node to probe it. Or if the neighbour node is AMN, then the SPX will flag the current node as danger.

By using the single point strategy, the SPX could get more information than the RPX. However, it is clear that the single point strategy does not give sufficient information to win all boards. If all of the remaining cells are neither AFN not AMN, then the single point strategy will not be able to give more information to the logical agent. If so, then the SPX will use a random method to choose the next cell to probe. So, it is possible to say that the SPX uses the single point strategy to reduce the chances of make new moves with random choices. However, if the single point strategy cannot give more information, then the SPX will make a random move just like RPX.

2 - 2 - 2. SATX

The SATX uses the SAT solver to get more information. Basically, the SAT solver is better than the single point strategy, since the SAT solver could solve some problems that the single point strategy cannot solve. Thus, it is possible to say that the SATX is the improved SPX.

To use the SAT solver, we need to build the KBU, and check the satisfiability of the CNF. Apparently, this uses more computational resources than the single point strategy. Thus, I designed my SATX to use both single point strategy and SAT solver. First, the SATX will try to make a move with single point strategy. If the SATX fails to get information with the single point strategy, then the SATX will use the SAT solver to get information. If the SAT solver also fails, then the SATX will use the random method to choose the next cell to probe.

As my SATX uses the single point strategy, I made the SATX class to be a child class of the SPX class, so that I could reuse the codes that I wrote for the SPX class. And I implemented the SAT solver by using the LogicNG and sat4j. First, I build the KBU, and use the LogicNG to convert the KBU to DIMACS CNF. Then, I use the sat4j to check the satisfiability of the KBU to choose the next cell.

2 - 3. Extension

I did 2 extensions for this coursework - 1) Add a new rule 2) The triangular map.

For the first extension, I add a new rule called “Multiple Lives”. Basically, the lives give the chance to logical agents to continue the game even if it probe the cell that contains the tornado. For example, if the user put 6 as a third command line argument, then the logical agent will have 6 lives. If the logical agent probe the cell that contains the tornado, then the program will decrease the remaining lives. In this case, the game will be over when the number of remaining lives is 0.

For the second extension, I made the logical agent to play with the triangular map. I implemented 4 triangular maps - T1, T2, T3 and T4. All logical agents could play the game with the triangular maps. The only difference between the hexagonal board (the original board) and the triangular board is the shape of the board. The nodes in the board have 6 neighbour nodes, and all valid cells contain either the number of neighboured danger cells or tornado.

3. Evaluation

In this section, I will evaluate the efficiency of the logical agents that I implemented.

The RPX uses the random method to choose the next cell. As the RPX chooses the node randomly, there is a possibility that the chosen cell contains the tornado. Therefore, it is possible to say the RPX is not a good logical agent, and it usually fails to win the game. To be fair, the RPX will not be able to win the game in general.

The SPX could win some games if the single point strategy provides sufficient information about the board. For some small and medium boards, the SPX could always win the game (i.e. S1, S2, M1, etc). However, it still makes several random choices. Especially, if the size of the board is large, the SPX uses many random choices.

The SATX could win some games that the SPX cannot win. For example, if the board is S4, then the SPX cannot win the game in general, however, the SATX could always win the game. Henceforth, it is possible to say that the SAT solver actually helps the SATX to reduce the number of random guesses. My SATX still makes some random guesses with some medium and large boards. However, it is clear that the SAT solver is actually better than the single point strategy.

4. Testing

My program requires at least 2 command line arguments, which are the name of the logical agent and the ID of the board that the agent will play with. If you do not input the suitable number of command line arguments, then the program will print out the error message just like below.

```
Usage: java -cp .:\* A2main <RPX|SPX|SATX> <ID> [any param]
```

If you use the invalid agent name as a first command line argument, then you will get the error message.

Command: java -cp .:* A2main RRX S1

```
Invalid Agent - RRX
```

Similarly, if the second command line argument is invalid, then you will also get the error message.

Command: java -cp .:* A2main SPX S0

```
Invalid ID - S0
```

Also, if you input the negative integer or some invalid value as a third command line argument, then you will get suitable error message.

Command: java -cp .:* A2main SPX S1 -12

```
The number of lifes should be an positive integer!
```

Command: java -cp .:* A2main SPX S1 “adasfdas”

```
Usage: java -cp .:\* A2main <RPX|SPX|SATX> <ID> <numberOfLifes>
```

If you pass suitable command line arguments, then you will be able to run the program properly.

Command: java -cp .:* A2main SPX S1

```
Single point strategy for hexagonal worlds (SPX)

Probe 0 0
Probe 2 2
Probe 0 1
Probe 1 1
Probe 2 1
Probe 3 1
Probe 4 1
Probe 2 0
Probe 1 0
Probe 3 0
Probe 4 2
Probe 3 2
Probe 4 3
Probe 1 2
Probe 0 2
Probe 1 3
Probe 0 3
Probe 2 3
Flag 0 4
```

```
Flag 0 4
Flag 1 4
Probe 2 4
Flag 3 3
Flag 3 4
Flag 4 0
Probe 4 4

Game end!
Result : Game won!
```

Command: java -cp .:* A2main SATX M4

```
Satisfiability Strategy for hexagonal worlds (SATX)

Probe 0 0
Probe 3 3
Probe 0 1
Probe 1 1
Probe 1 2
Probe 0 2
Probe 1 0
```

```
(@RESERVED_CNF_0 & ~@RESERVED_CNF_1) | (~@RESERVED_CNF_0 & @RESERVED_CNF_1)
(@RESERVED_CNF_1 & ~@RESERVED_CNF_2) | (~@RESERVED_CNF_1 & @RESERVED_CNF_2)
(@RESERVED_CNF_3 & ~@RESERVED_CNF_4) | (~@RESERVED_CNF_3 & @RESERVED_CNF_4)
(@RESERVED_CNF_2 & @RESERVED_CNF_5 & ~@RESERVED_CNF_3) | (@RESERVED_CNF_2 & ~@RESERVED_CNF_5 & @RESERVED_CNF_3) | (~@RESERVED_CNF_2 & @RE
SERVED_CNF_5 & @RESERVED_CNF_3)
(@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & @R
ESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & @RESE
RVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & @RESERVE
D_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & @RESERVED_C
NF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & @RESERVED_CNF_
9)
((@RESERVED_CNF_0 & ~@RESERVED_CNF_1) | (~@RESERVED_CNF_0 & @RESERVED_CNF_1)) & ((@RESERVED_CNF_1 & ~@RESERVED_CNF_2) | (~@RESERVED_CNF_1
 & @RESERVED_CNF_2)) & ((@RESERVED_CNF_3 & ~@RESERVED_CNF_4) | (~@RESERVED_CNF_3 & @RESERVED_CNF_4)) & ((@RESERVED_CNF_2 & @RESERVED_CNF_
5 & ~@RESERVED_CNF_3) | (@RESERVED_CNF_2 & ~@RESERVED_CNF_5 & @RESERVED_CNF_3) | (~@RESERVED_CNF_2 & @RESERVED_CNF_5 & @RESERVED_CNF_3))
& ((@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 &
 @RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & @R
ESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & @RESE
RVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & @RESERVE
D_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & @RESERVED_C
NF_9))
```

```
Unique Literals:
1: @RESERVED_CNF_0 = (2, 0)
2: @RESERVED_CNF_1 = (2, 1)
3: @RESERVED_CNF_2 = (2, 2)
4: @RESERVED_CNF_3 = (1, 3)
5: @RESERVED_CNF_4 = (0, 3)
6: @RESERVED_CNF_5 = (2, 3)
7: @RESERVED_CNF_6 = (4, 3)
8: @RESERVED_CNF_7 = (3, 2)
9: @RESERVED_CNF_8 = (4, 4)
10: @RESERVED_CNF_9 = (3, 4)
```

```
Before CNF:
((@RESERVED_CNF_0 & ~@RESERVED_CNF_1) | (~@RESERVED_CNF_0 & @RESERVED_CNF_1)) & ((@RESERVED_CNF_1 & ~@RESERVED_CNF_2) | (~@RESERVED_CNF_1
 & @RESERVED_CNF_2)) & ((@RESERVED_CNF_3 & ~@RESERVED_CNF_4) | (~@RESERVED_CNF_3 & @RESERVED_CNF_4)) & ((@RESERVED_CNF_2 & @RESERVED_CNF_
5 & ~@RESERVED_CNF_3) | (@RESERVED_CNF_2 & ~@RESERVED_CNF_5 & @RESERVED_CNF_3) | (~@RESERVED_CNF_2 & @RESERVED_CNF_5 & @RESERVED_CNF_3))
& ((@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 &
 @RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & @R
ESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & @RESE
RVED_CNF_7 & ~@RESERVED_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & @RESERVE
D_CNF_8 & ~@RESERVED_CNF_9) | (~@RESERVED_CNF_5 & ~@RESERVED_CNF_6 & ~@RESERVED_CNF_2 & ~@RESERVED_CNF_7 & ~@RESERVED_CNF_8 & @RESERVED_C
NF_9))
CNF KBU:
(@RESERVED_CNF_1 | @RESERVED_CNF_0) & (~@RESERVED_CNF_0 | ~@RESERVED_CNF_1) & (@RESERVED_CNF_2 | @RESERVED_CNF_1) & (~@RESERVED_CNF_1 | ~
@RESERVED_CNF_2) & (@RESERVED_CNF_4 | @RESERVED_CNF_3) & (~@RESERVED_CNF_3 | ~@RESERVED_CNF_4) & (@RESERVED_CNF_2 | @RESERVED_CNF_5) & (@
RESERVED_CNF_3 | @RESERVED_CNF_2) & (@RESERVED_CNF_3 | ~@RESERVED_CNF_5 | @RESERVED_CNF_2) & (@RESERVED_CNF_5 | @RESERVED_CNF_3 | @RESERV
ED_CNF_2) & (~@RESERVED_CNF_2 | @RESERVED_CNF_3 | @RESERVED_CNF_5) & (@RESERVED_CNF_3 | @RESERVED_CNF_5) & (@RESERVED_CNF_5 | @RESERVED_C
NF_2 | ~@RESERVED_CNF_3) & (~@RESERVED_CNF_2 | ~@RESERVED_CNF_5 | ~@RESERVED_CNF_3) & (~@RESERVED_CNF_0 | @RESERVED_CNF_5) & ~@RESERVED_C
NF_0 & (~@RESERVED_CNF_0 | ~@RESERVED_CNF_2) & (~@RESERVED_CNF_0 | ~@RESERVED_CNF_7) & (~@RESERVED_CNF_0 | ~@RESERVED_CNF_8) & (~@RESERVE
D_CNF_0 | ~@RESERVED_CNF_9) & (~@RESERVED_CNF_1 | ~@RESERVED_CNF_5) & (~@RESERVED_CNF_1 | ~@RESERVED_CNF_7) & (~@RESERVED_CNF_1 | ~@RESER
VED_CNF_8) & (~@RESERVED_CNF_1 | ~@RESERVED_CNF_9) & (@RESERVED_CNF_1 | @RESERVED_CNF_5 | @RESERVED_CNF_2 | @RESERVED_CNF_7 | @RESERVED_C
NF_8 | @RESERVED_CNF_9) & (~@RESERVED_CNF_5 | ~@RESERVED_CNF_2) & ~@RESERVED_CNF_2 & (~@RESERVED_CNF_7 | ~@RESERVED_CNF_2) & (~@RESERVED_
CNF_8 | ~@RESERVED_CNF_2) & (~@RESERVED_CNF_9 | ~@RESERVED_CNF_2) & (~@RESERVED_CNF_5 | ~@RESERVED_CNF_3) & ~@RESERVED_CNF_3 & (~@RESERVE
D_CNF_3 | ~@RESERVED_CNF_2) & (~@RESERVED_CNF_3 | @RESERVED_CNF_7) & (~@RESERVED_CNF_3 | ~@RESERVED_CNF_8) & (~@RESERVED_CNF_3 | ~@RESERV
ED_CNF_9) & (~@RESERVED_CNF_4 | ~@RESERVED_CNF_5) & ~@RESERVED_CNF_4 & (~@RESERVED_CNF_4 | ~@RESERVED_CNF_2) & (~@RESERVED_CNF_4 | ~@RESE
RVED_CNF_7) & (~@RESERVED_CNF_4 | @RESERVED_CNF_8) & (~@RESERVED_CNF_4 | ~@RESERVED_CNF_9) & ~@RESERVED_CNF_5 & (~@RESERVED_CNF_5 | ~@RES
ERVED_CNF_7) & (~@RESERVED_CNF_5 | ~@RESERVED_CNF_8) & (~@RESERVED_CNF_5 | @RESERVED_CNF_9) & (@RESERVED_CNF_0 | @RESERVED_CNF_1 | @RESER
VED_CNF_2 | @RESERVED_CNF_3 | @RESERVED_CNF_4 | @RESERVED_CNF_5)
```

```
Flag 1 3
Probe 1 4
Flag 1 5
Probe 1 6
Probe 0 3
Probe 0 4
Flag 0 5
Probe 0 6
Probe 6 0

Game end!
Result : Game won!
```

Command: java -cp .:* A2main RPX TEST0

```
Randome Probe (RPX)


The coordinate (0, 4) contains the tornado!
Game over!
```

Command: java -cp .:* A2main RPX TEST0 2

```
Randome Probe (RPX)


The coordinate (0, 4) contains the tornado!
Multiple life mode - auto flag the current coordinate
Flag 0 4
Remaining lifes = 1

Probe 2 2
Probe 3 1
Probe 1 3
Probe 1 4
Probe 4 1
Probe 3 4
Probe 0 0

The coordinate (2, 4) contains the tornado!
Game over!
```

Command: java -cp .:* A2main SPX T2

```
Single point strategy for hexagonal worlds (SPX)

Probe 0 0
Probe 2 2
Probe 1 1
Probe 1 2
Probe 2 3
Probe 3 3
Probe 4 4
Probe 3 4
Flag 2 4
Flag 2 4
Flag 1 3
Flag 0 1
Probe 0 2
Probe 0 3
Flag 0 4
Flag 1 4

Game end!
Result : Game won!
```