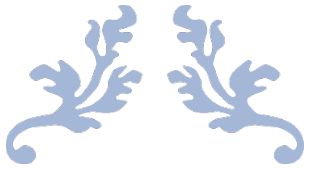


CS5011 A2

200010781

MARCH 9, 2022

Contents

1 Introduction

This report describes the second assignment for CS5011 module. General information can be seen in the Table 1 below.

Table 1 General information of the assignment

Assignment Number	Assignment 2
Date of Submission	9 March 2022
Student ID	200010781
Word Count	1995

There are 3 ferry agents in this assignment, including basic, intermediate, and advanced ferry agents. The completion of each agent is described as Table 2 below.

Table 2 Completion of each part

Basic	Basic Agent	attempted, fully working
Beginner	SPS	attempted, fully working
Intermediate	DNF encoding	attempted, fully working
	CND encoding	attempted, fully working

About running the source code, several steps are needed.

1. Enter the terminal.
2. Change to the CS5011_A2 directory.
3. Compile all source code

```
javac *.java help/*.java agent/*.java startCode/*.java -cp .;../libs/*
```

4. Run the source code

   java -cp .;../libs/* A2main <P1|P2|P3|P4> <ID> [verbose]

   The “-cp .;../libs/*” is used to include the libraries needed in our projects

   For example, if I want to use SPS to player the mine sweeper game of TEST1, I should use

   java -cp .;../libs/* A2main P2 TEST1

2 Design and Implementation

2.1 Problem Overview

This assignment focuses on playing the obscured sweeper game through different logic agents. In this assignment, the PEAS model is described as Table 3 below.

Table 3 PEAS model for assignment 2

Agent	**Performance Measure**	**Environment**	**Actuator**	**Sensor**
Logic Agents for Obscured Sweeper Game	Cell Probed, Game Success	Board	Probe a cell, Flag a cell	Logic Strategies

2.2 System Architecture

All source code can be found in the src directory. The main function is addressed in A2main.java file which locates in src. There are three packages in the src directory.

- **startCode**
  - ✧ World class provided by lecturer locates in this package.
  - ✧ GameHandler class helps to handle the game. It knows everything about the game board and provide some information for agents.
- **agent**

  There are four classes in this package, named P1, P2, P3 and P4.
  - ✧ P1 class focuses for the basic obscured sweeper game, which probe covered cells in order. Once probed a mine, the game is over.
  - ✧ P2 class uses single point reasoning strategy to play the games. It uses AMN and AFN to determine whether a cell is safe or not.
  - ✧ P3 class uses SAT Solver by DNF encoding to determine whether a cell contains a mine or not. Since we need to use SPS to play the game at first, P3 class inherits from P2 class. It calls the constructor of P2 usung SPS at first. If SPS cannot solve the game, it will pass the agent view to P3 agent. P3 agent builds knowledge base in DNF format at first. Then use sat solver from LogicNG library to decide a cell is safe or unsage.
  - ✧ P4 class uses CNF encoding to solve the problem. It uses P2 and P3 agent at first. Therefore, P4 class inherits from P3 class and call its constructor at first. When both cannot solve the problem, the final unsolved agent view will be passed to P4 agent. Similar as P3 agent, P4 agent also builds knowledge base at first, but in CNF format. Then it uses SAT4J Core library to solve the satisfiability problems.
- **help**
  - ✧ Help class exists some Auxiliary methods.

2.3 Implementation

2.3.1 Basic (P1) Agent

This agent probes cells in the board in order, from left to right and from top to bottom. If probe a covered cell whose clue is 0, the uncoverNeighbor() method is called to open all neighbours in this cell. If the agent probes a cell with mine, the agent fails the game. If the user input a “verbose” parameter when running, the agent will print each step during probing. When the agent probed all the save cells, it wins the game and the agent alive.

2.3.2 Beginner (P2) Agent

This agent uses single point strategy. It probes covered cell through AMN and AFN strategies from lectures. The agent checks the neighbours of the covered cell. If a cell satisfies the AMN rule (Num(covered) = clue – Num(flag)), the agent will mark a danger. If a cell satisfies the AFN rule (clue= Num(flag)), the agent will uncover the cell. If probe a covered cell whose clue is 0, the uncoverNeighbor() method is called to open all neighbours in this cell. The agent uses checkUncoverNeighbor() to count the number of flagged cell and covered cell of neighbours. I use several variables to record the number of probed cells and flagged cells. If the agent probes all the safe cells and flagged all the danger cells, the agent wins the game.

2.3.3 Intermediate (P3) Agent

This agent uses SAT solver with DNF coding to play the game. In the constructor of P3, it calls its parent constructor using super(). If P2 agent cannot solve the game, it will pass its agent view to P3. The P3 agent calls buildKBU() to build the knowledge base in DNF encoding format at first. In this assignment, I use the coordinates of cells to represent the cells in KBU. For example, a cell locates in (1,2), then I will represent it in KBU using “12”. cellFormula() method is used to construct the formula of each cell. An important thing needs to be solved is getting permutation of covered cells to construct formulas. Permutation() method is used to address this problem with recursion and stack. All permutation results will be stored into a two-dimensional array named possibleDanger, which will be used to construct the cell formulas. Same as P2 agent, once the agent probes all the safe cells and

flagged all the danger cells, it wins the game.

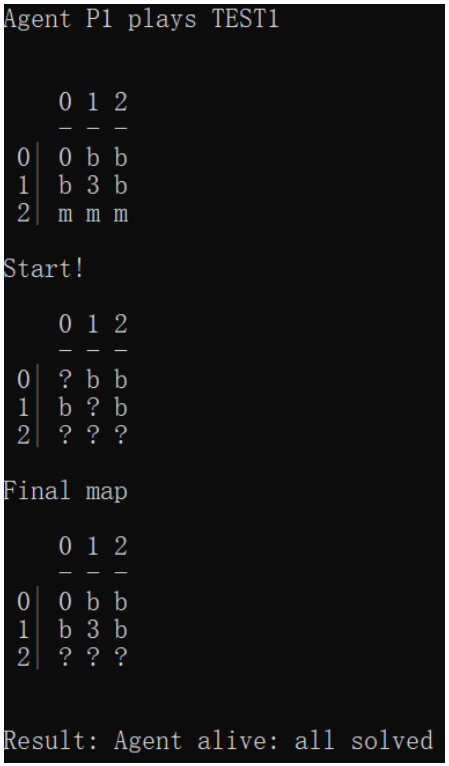
2.3.4 Intermediate2 (P4) Agent

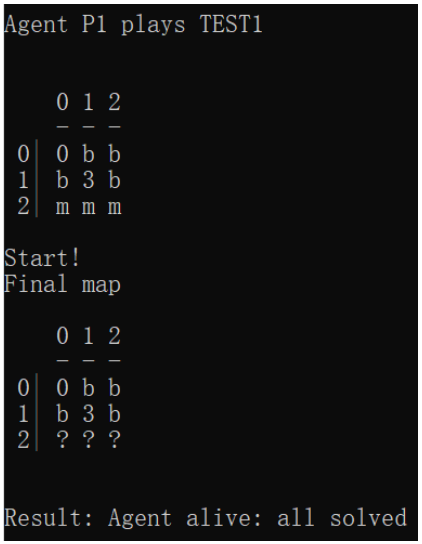
This agent uses SAT4J Core library to determine a cell is safe or unsafe. In P4 constructor, it calls P3 constructor using super(), where P3 will call P2 constructor using super() as well. If P2 and P3 agents cannot solve the game, final agent view will be passed to P4. P4 agent calls buildKBUCNF() to build the knowledge base, but in CNF encoding format. cellFormulaCNF() method is also used to construct the formula of each cell, but aiming at creating CNF format. In cellFormulaCNF() method, it uses two permutation() methods to create the cell formula (exact k dangers = at most k danger + at least k danger). The first permutation is used to construct the permutation of at most k dangers, while the second is used to construct the at least k danger permutation. After who permutations are constructed, they will be added in to clauses ArrayList, which will be used to build DIMACS format formula for further solver to determine. Same as previous agents, once the agent probes all the safe cells and flagged all the danger cells, it wins the game.

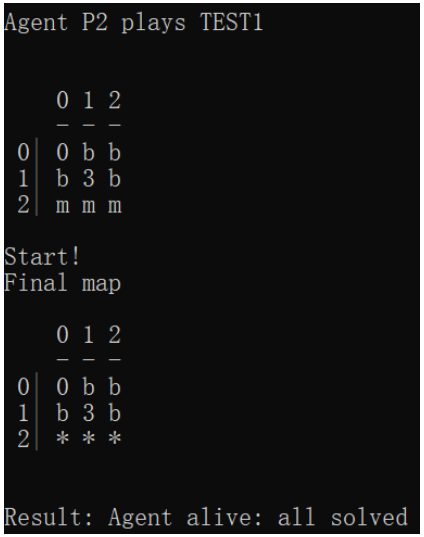
3 Test Summary

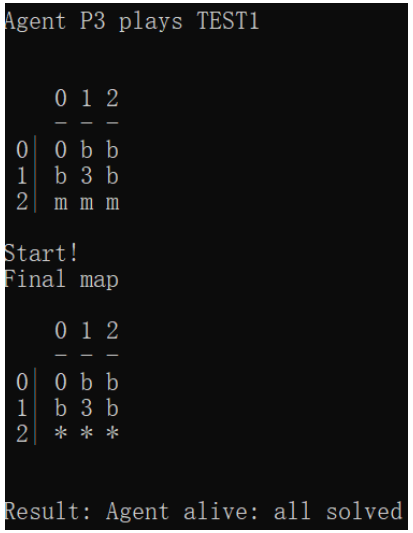
In this assignment, I tested all test cases and select two cases in each part (TEST, SMALL, MEDIUM, and LARGE) using different agents.

I select TEST1 and TEST3 two cases in the TEST parts. The outputs of four agents can be shown as Figure 1 and Figure 2 below. The first picture uses P1 with verbose parameter to print each step when playing the game. For this test case, SPS (P2 agent) can win the game successfully. Therefore, the P3 and P4 agent can win the game easily using P2 strategy. In TEST3 test cases, P1 agent die when probing the (2,1) cell, which contains a mine. P2 agent using SPS cannot solve this game, and it leaves three covered cells because it cannot determine their safety. P3 agent using DNF encoding SAT solver finished this game successfully, which leads that P4 agent can solve this test case before using CNF encoding.









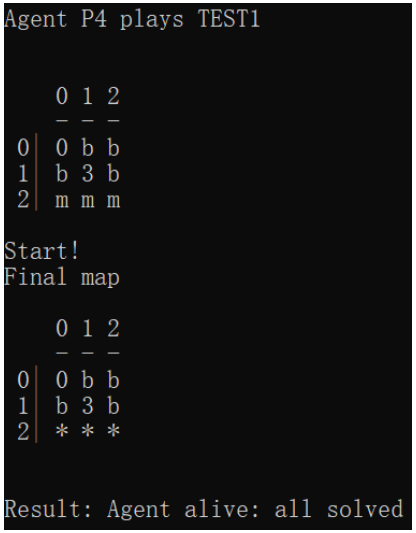
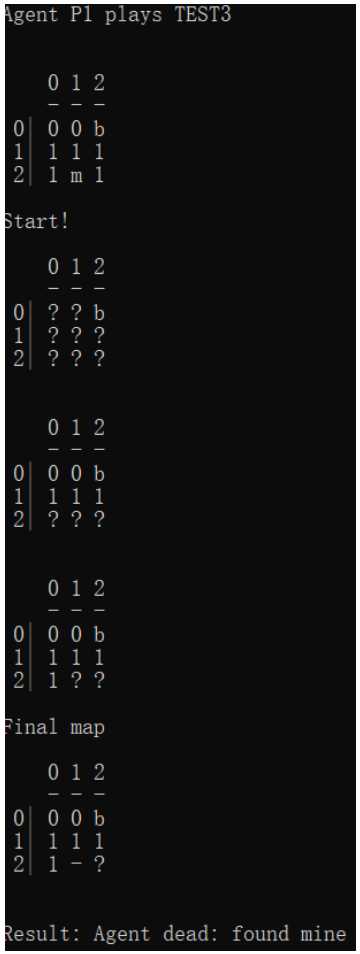
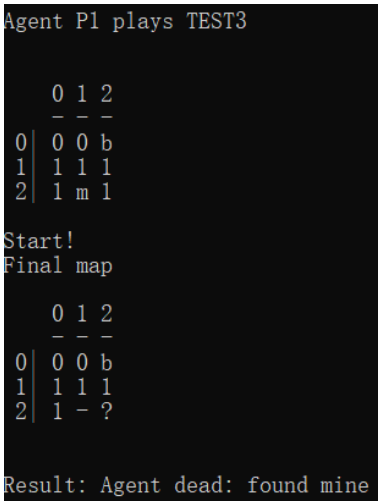
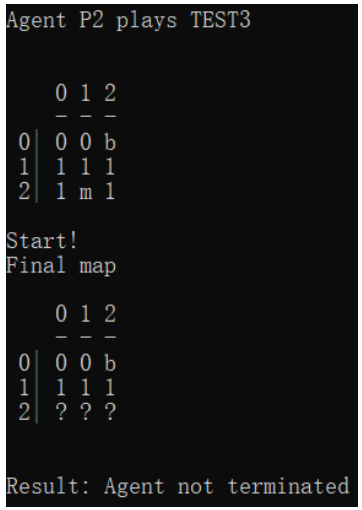
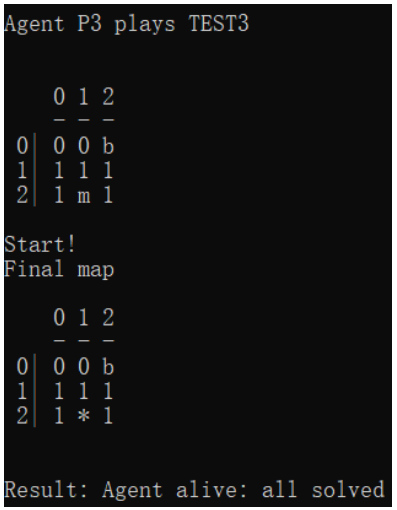


Figure 1 Different outputs for TEST1









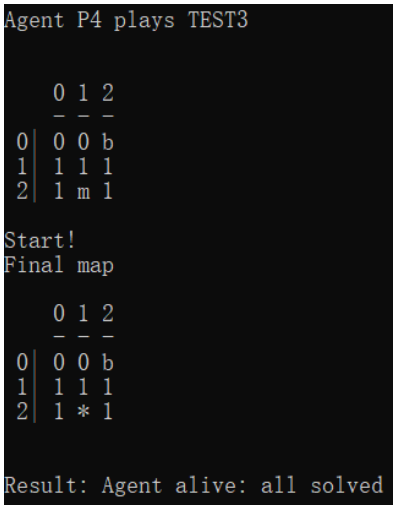


Figure 2 Different outputs for TEST3

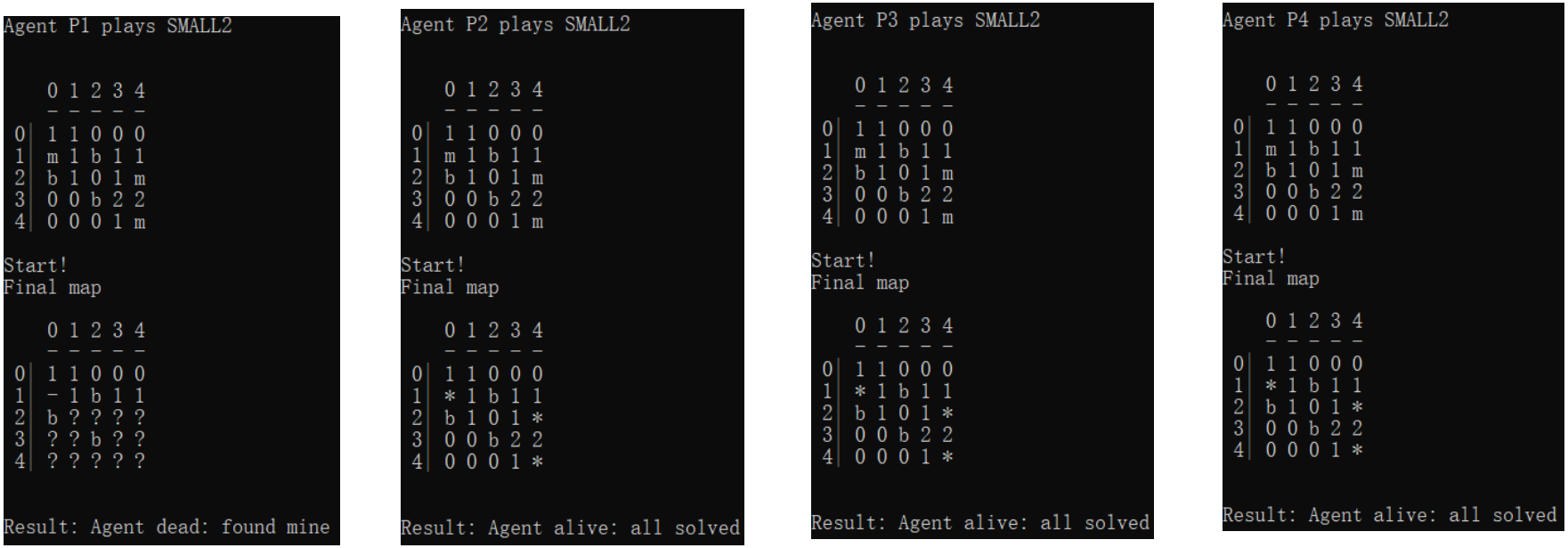


Figure 3 Different outputs for SMALL2

In SMALL part, I select SMALL2 and SMALL5 two cases. The outputs of four agents can be shown as Figure 3 and Figure 4. Since the output steps in P1 when using verbose parameter agent are more than before, it is hard to show them all in the report. Therefore, I will not include the screenshots in the report. I suggest running the source code to check its correctness. In SMALL2 cases, P1 agent died in the (1,0) cell which contains a mine. P2 agent finished the game successfully. Therefore, P3 and P4 solve this game correctly as well.

In SMALL5 test case, P1 agent died in the (2,2) cell which contains a mine. P2 agent using SPS cannot win this game. Fortunately, P3 agent using DNF encoding win the game successfully, as well as P4 agent.

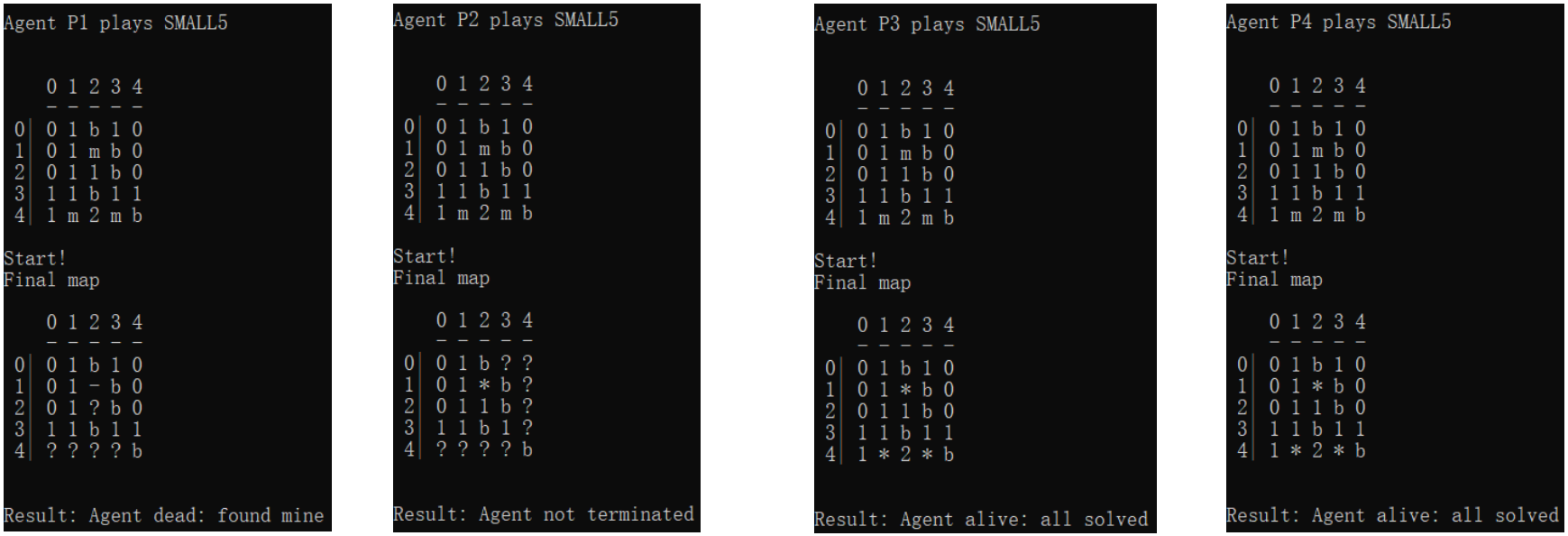


Figure 4 Different outputs for SMALL5

I select MEDIUM5 and MEDIUM6 two cases in this part. In MEDIUM. In MEDIUM5, as shown in Figure 5, P1 agent died in (0,5) cell, while P2 agent leaves 39 cells covered. P3 agent finishes the game successfully, and same to P4 agent. In MEDIUM6, as shown in Figure 6, all agents cannot finish the game. P1 died in (0,4) cell, while P2 left 6 cells covered and P3 and P4 left 2 cells covered.

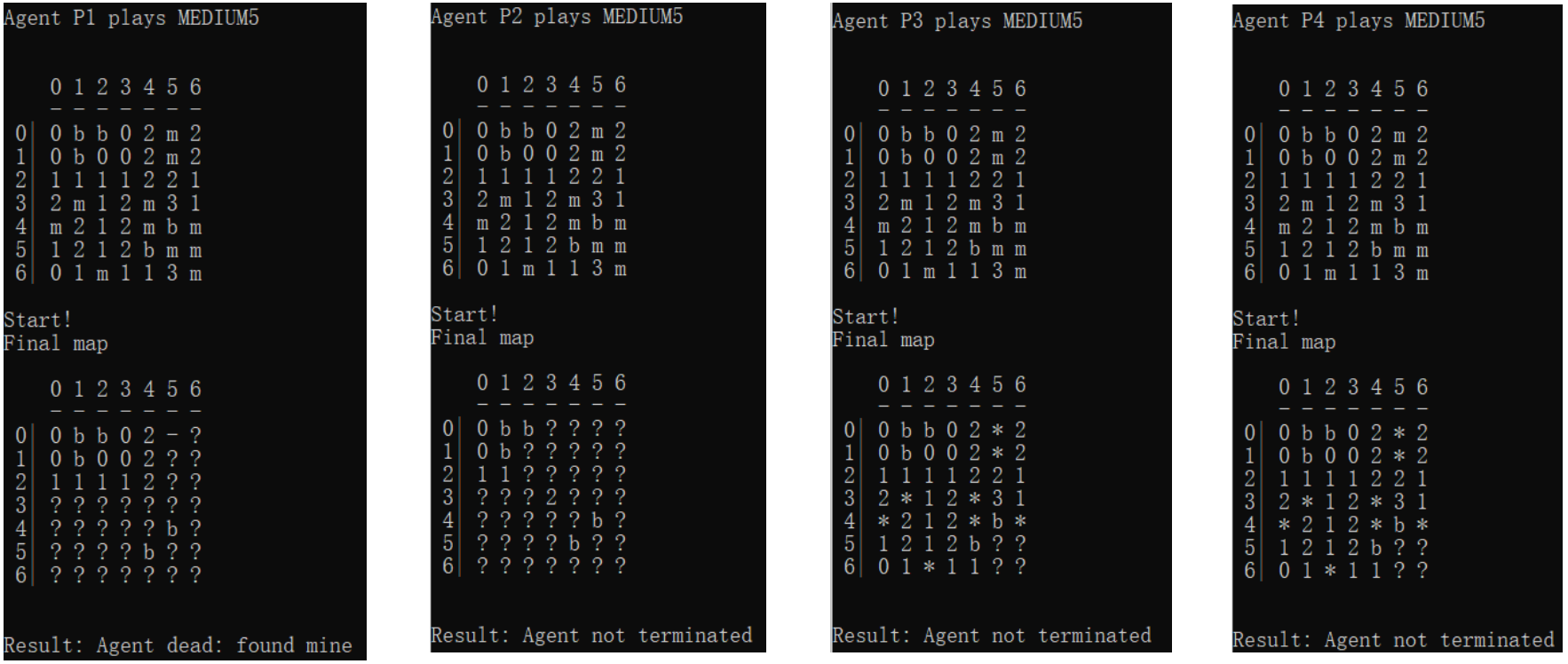
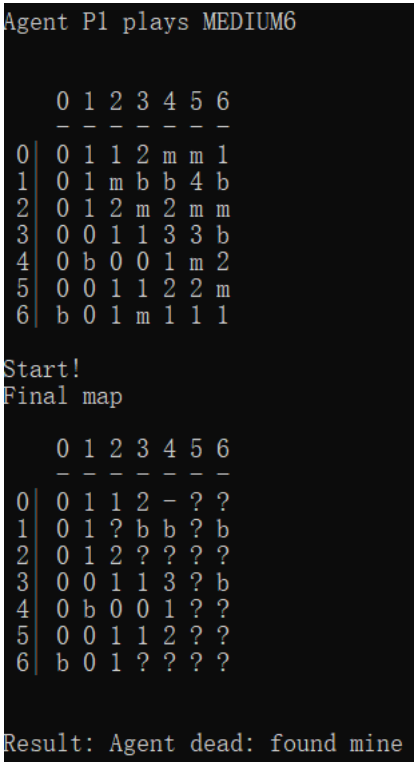
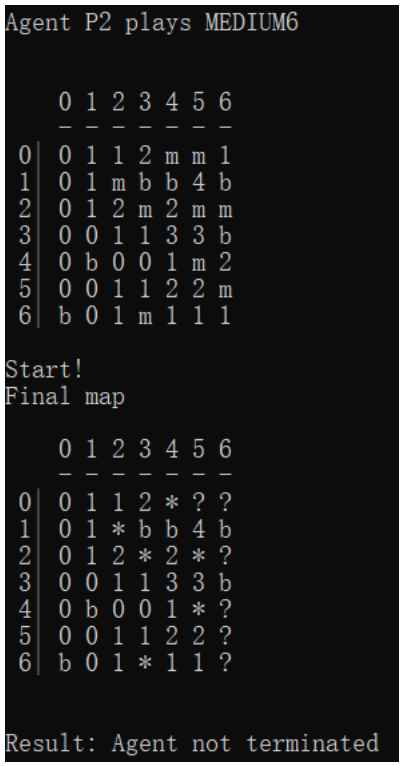
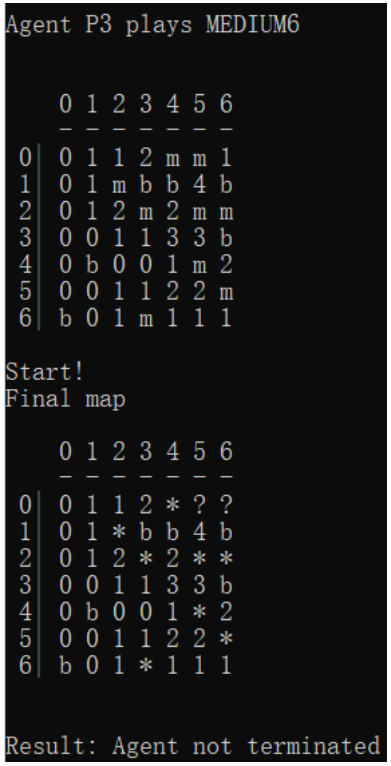


Figure 5 Different outputs for MEDIUM5







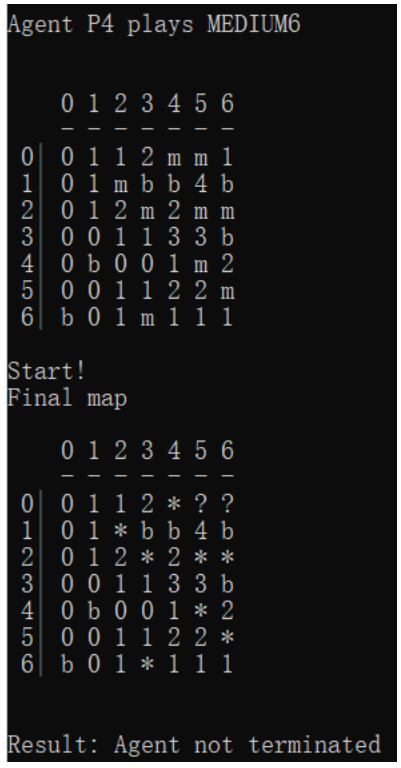


Figure 6 Different outputs for MEDIUM6

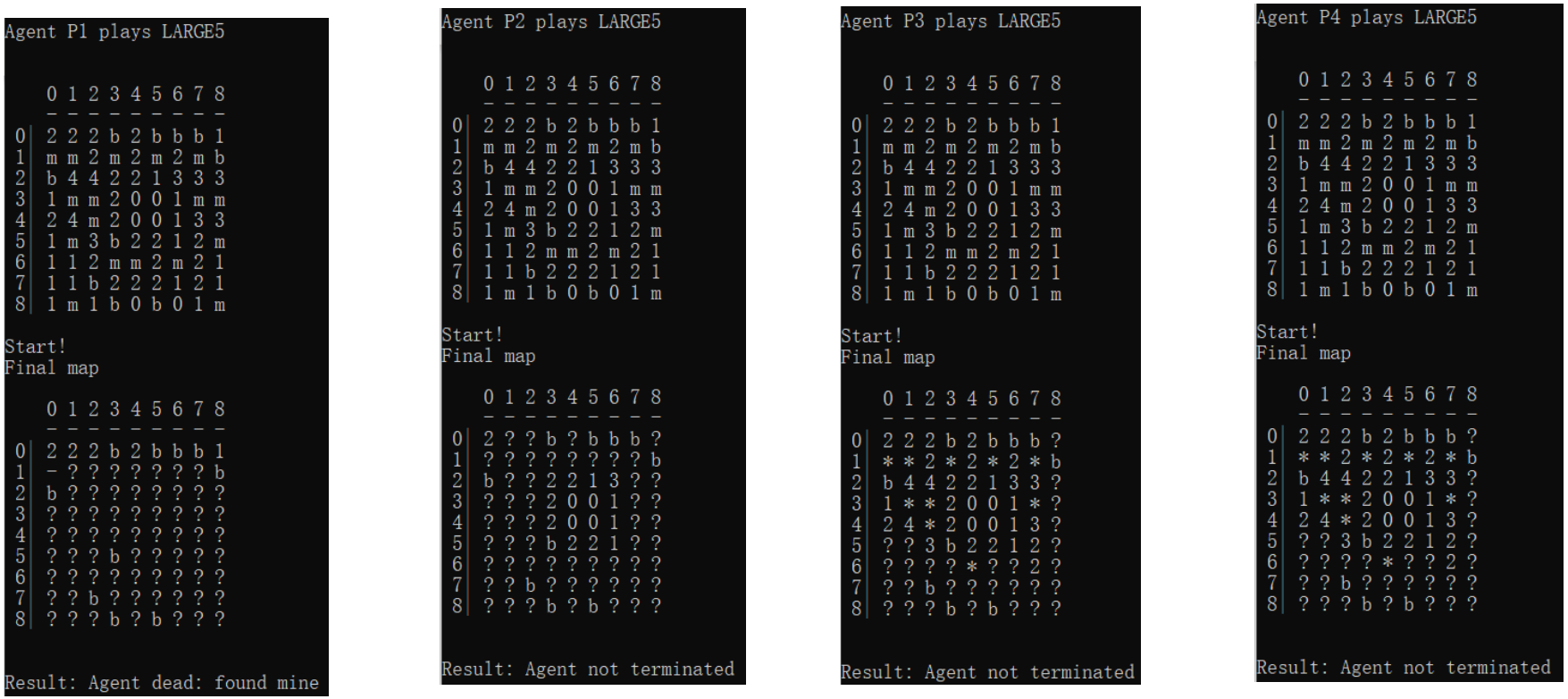
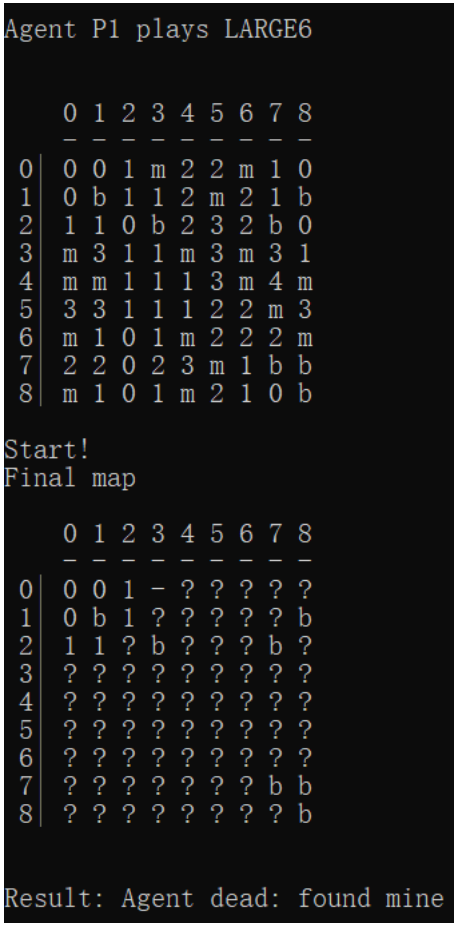
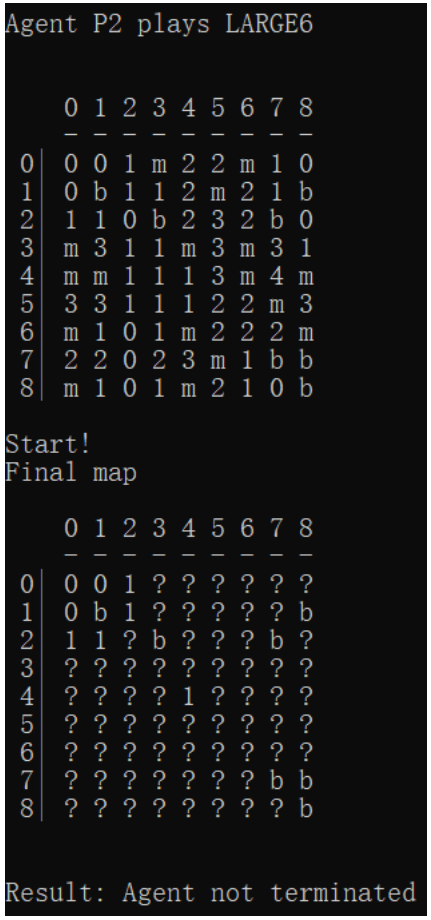
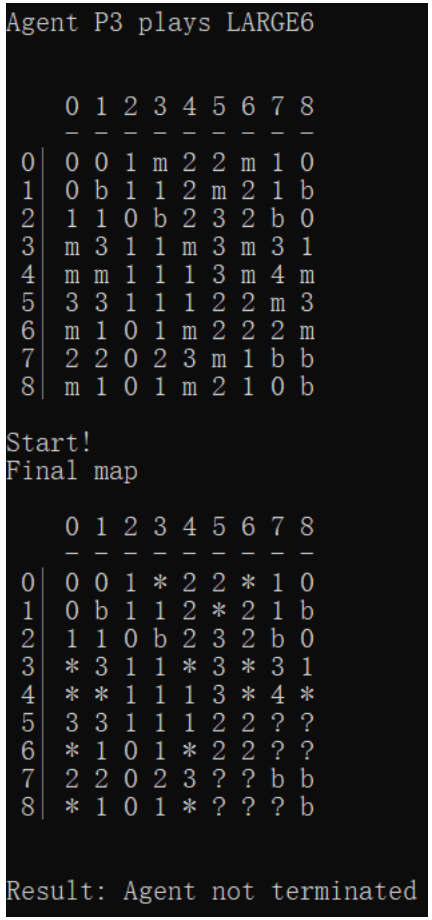


Figure 7 Different outputs for LARGE5

In LARGE cases, I select LARGE5 and LARGE6 two test cases, as shown in Figure 7 and Figure 8. For LARGE 5 case, all agents cannot finish the game, but it is clear that the number of covered cells is decreasing from the P1 to P4. Similar situation can be seen in LARGE6. P1 died in the cell (0,3), and other three agents cannot finish the game successfully.







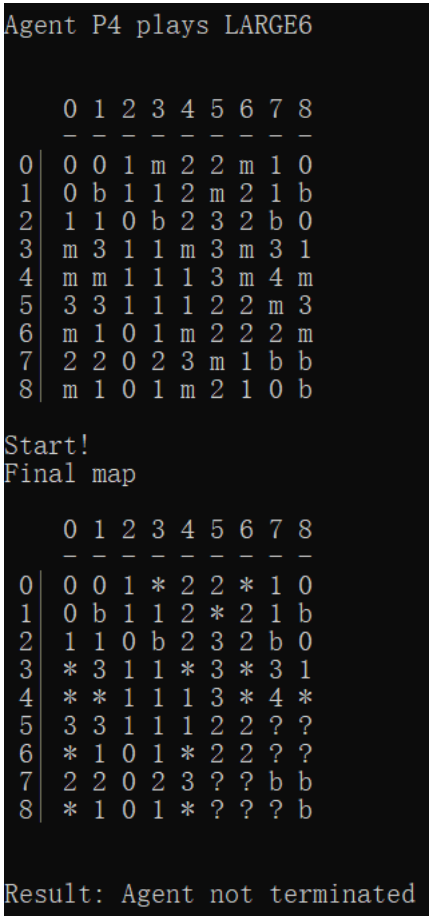


Figure 8 Different outputs in LARGE6

4 Evaluation and Conclusion

I tested all test cases and recorded the statues and the cells left in each case using different agents. I also record the failure place of P1 when probing a mine. The results can be seen in the table below.

From vertical perspective, intermediate agents works better than beginner and basic agents. In all test cases, intermediate agents play the game successfully in 8 test cases, while beginner agent wins the game only four times and basic agent fails all the test cases. Besides, when the board size grows large, such as in LARGE test cases, more cells tend to be left in all logic agents. For example, the highest percentage of left cells in beginner agent in SMALL cases is 72%. The number increases to 79.6% in MEDIUM cases. In LARGE cases, it rises to 81.5%. Moreover, all agents cannot win the game in LARGE cases. This means that logic is not omnipotent, but it indeed can provide some help since intermediate left less cells to be probed than other two from the table.

From horizontal perspective, intermediate agents probed more cells than basic and beginner agents. For example, in MEDIUM10 test case, there are 40 cells are still covered after using basic agent and 10 cells left after using beginner agent, while intermediate agent wins the game, which means it probed all safe cells and flagged all danger cells. Beginner agent probed more cells than basic agent in most cases. In some situations, basic agent can probe more cells than beginner. These situations happen depending on the location of the first mine. If the first mine appears in a later order position, the basic agent can probe more cells.

It is worth to notice that P3 and P4 agent get the total same results. This means no matter CNF and DNF encoding will not affect the result we get, but only different representing formats.

Table 4 The results of testing all cases using different agents

CASES	BASIC(P1)			BEGINNER(P2)			INTERMEDIATE(P3)			INTERMEDIATE -2(P4)		
	STATUS (die in)	Left cells (Num and %)		STATUS	Left cells (Num and %)		STATUS	Left cells (Num and %)		STATUS	Left cells (Num and %)	
SMALL1	DEAD (1,3)	8	32%	Not Terminate	6	24%	Alive	-	-	Alive	-	-
SMALL2	DEAD (1,0)	13	52%	Alive	-	-	Alive	-	-	Alive	-	-
SMALL3	DEAD (0,2)	18	72%	Not Terminate	18	72%	Not Terminate	16	64%	Not Terminate	16	64%
SMALL4	DEAD (0,4)	12	48%	Not Terminate	8	32%	Not Terminate	8	32%	Not Terminate	8	32%
SMALL5	DEAD (1,2)	5	20%	Not Terminate	9	36%	Alive	-	-	Alive	-	-
SMALL6	DEAD (2,0)	4	16%	Alive	-	-	Alive	-	-	Alive	-	-
SMALL7	DEAD (0,1)	19	76%	Alive	-	-	Alive	-	-	Alive	-	-
SMALL8	DEAD (0,2)	14	56%	Not Terminate	2	8%	Not Terminate	2	8%	Not Terminate	2	8%
SMALL9	DEAD (3,0)	6	24%	Not Terminate	5	20%	Not Terminate	5	20%	Not Terminate	5	20%
SMALL10	DEAD (0,3)	5	20%	Not Terminate	1	4%	Not Terminate	1	4%	Not Terminate	1	4%
CASES	Basic(P1)			Beginner(P2)			Intermediate(P3)			Intermediate -2(P4)		
	STATUS (die in)	Left cells (Num and %)		STATUS	Left cells (Num and %)		STATUS	Left cells (Num and %)		STATUS	Left cells (Num and %)	
MEDIUM1	DEAD (1,5)	28	57.1%	Not Terminate	22	44.9%	Not Terminate	21	42.9%	Not Terminate	21	42.9%
MEDIUM2	DEAD (2,5)	23	46.9%	Not Terminate	16	32.7%	Not Terminate	16	32.7%	Not Terminate	16	32.7%
MEDIUM3	DEAD (0,5)	34	69.4%	Not Terminate	37	75.5%	Alive	-	-	Alive	-	-
MEDIUM4	DEAD (2,3)	26	53.1%	Not Terminate	20	40.8%	Not Terminate	18	36.7%	Not Terminate	18	36.7%
MEDIUM5	DEAD (0,5)	31	63.3%	Not Terminate	39	79.6%	Not Terminate	4	8.2%	Not Terminate	4	8.2%
MEDIUM6	DEAD (0,4)	17	34.7%	Not Terminate	6	12.2%	Not Terminate	2	4.1%	Not Terminate	2	4.1%
MEDIUM7	DEAD (0,4)	31	63.3%	Not Terminate	31	63.3%	Not Terminate	6	12.2%	Not Terminate	6	12.2%

MEDIUM8	DEAD (0,6)	28	57.1%	Not Terminate	29	59.2%	Not Terminate	25	51.0%	Not Terminate	25	51.0%
MEDIUM9	DEAD (1,4)	32	65.3%	Alive	-	-	Alive	-	-	Alive	-	-
MEDIUM10	DEAD (0,3)	40	81.6%	Not Terminate	10	20.4%	Alive	-	-	Alive	-	-
CASES	**Basic(P1)**			**Beginner(P2)**			**Intermediate(P3)**			**Intermediate -2(P4)**		
	STATUS (die in)	**Left cells (Num and %)**		**STATUS**	**Left cells (Num and %)**		**STATUS**	**Left cells (Num and %)**		**STATUS**	**Left cells (Num and %)**	
LARGE1	DEAD (0,5)	60	74.1%	Not Terminate	63	77.8%	Not Terminate	59	72.8%	Not Terminate	59	72.8%
LARGE2	DEAD (0,3)	59	72.8%	Not Terminate	53	65.4%	Not Terminate	34	42.0%	Not Terminate	34	42.0%
LARGE3	DEAD (0,2)	53	65.4%	Not Terminate	11	13.6%	Not Terminate	11	13.6%	Not Terminate	11	13.6%
LARGE4	DEAD (0,3)	65	80.3%	Not Terminate	66	81.5%	Not Terminate	60	74.1%	Not Terminate	60	74.1%
LARGE5	DEAD (1,0)	65	80.3%	Not Terminate	55	67.9%	Not Terminate	29	35.8%	Not Terminate	29	35.8%
LARGE6	DEAD (0,3)	66	81.5%	Not Terminate	66	81.5%	Not Terminate	9	11.1%	Not Terminate	9	11.1%
LARGE7	DEAD (0,5)	65	80.3%	Not Terminate	40	49.4%	Not Terminate	6	7.4%	Not Terminate	6	7.4%
LARGE8	DEAD (0,7)	60	74.1%	Not Terminate	64	79.1%	Not Terminate	19	23.5%	Not Terminate	19	23.5%
LARGE9	DEAD (0,5)	38	46.9%	Not Terminate	20	24.7%	Not Terminate	20	24.7%	Not Terminate	20	24.7%
LARGE10	DEAD (0,4)	71	87.7%	Not Terminate	10	12.4%	Not Terminate	9	11.1%	Not Terminate	9	11.1%

In conclusion, logic agents can provide some help when playing the MineSweeper game. Although they are not omnipotent, but they indeed help to play the game correctly in most cases. Although all agents cannot win the game in LARGE test cases, intermediate agents indeed leave a smaller number of unknown cells than other two agents. Among all the four agents, intermediate agents work best, either in DNF encoding or in CNF encoding format. Beginner agent using single point strategy performs better than basic agent. This means that using logic is helpful to some extends