CS5011 Practical 3: Logic - Tornado Sweeper

Student ID: 220016150
Word Count: 2057

1. INTRODUCTION

1.1 Checklist of parts implemented

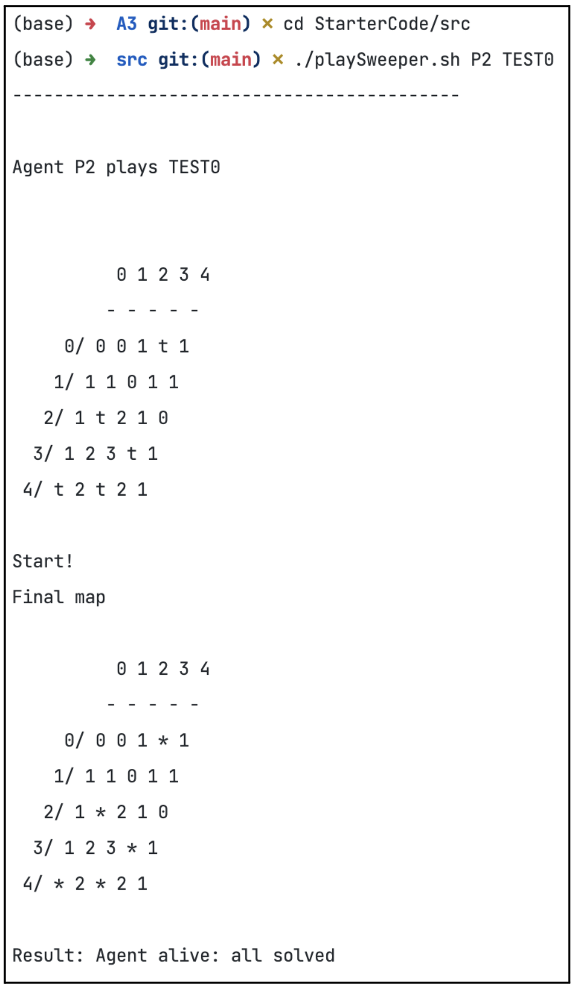
- Part 1: attempted, fully working
- Part 2: attempted, fully working
- Part 3: attempted, fully working
- Part 4: not attempted
- Part 5: not attempted

1.2 Compiling and running instructions

i. Change the directory into the `StarterCode/src` folder and use the following command

```
./playSweeper.sh <P1|P2|P3|P4> <ID> [verbose] [<any other param>]
```

The playSweeper is a shell script to import the external jar files and compile all the java code files and run the class file passing the arguments specified by the user.
<P1|P2|P3|P4> : the part type you want to execute
<ID>: The world you want the agent to solve
<verbose> (optional) to print the game state at different iterations



```
(base) ➜  A3 git:(main) ✗ cd StarterCode/src
(base) ➜  src git:(main) ✗ ./playSweeper.sh P2 TEST0
-------------------------------------------


Agent P2 plays TEST0



         0 1 2 3 4
        - - - - -
    0/ 0 0 1 t 1
   1/ 1 1 0 1 1
  2/ 1 t 2 1 0
 3/ 1 2 3 t 1
4/ t 2 t 2 1


Start!
Final map


         0 1 2 3 4
        - - - - -
    0/ 0 0 1 * 1
   1/ 1 1 0 1 1
  2/ 1 * 2 1 0
 3/ 1 2 3 * 1
4/ * 2 * 2 1


Result: Agent alive: all solved
```

(figure 1: Running the program and outcome)

2. DESIGN & IMPLEMENTATION

2.1 PEAS

The overview of the design of the Tornado Sweeper and the PEAS model for the problem is as under:

Performance:
- The agent is expected to probe as many cells as possible to reveal the locations of the tornadoes on the game board by avoiding probing cells that are likely to contain tornados and instead using flags to mark those cells for later probing.
- The agent needs to identify all the tornados on the board within a reasonable number of moves.

Environment:
- The environment is a rhombus grid representing the game board, with each cell being hexagonal in shape and can be probed or flagged by the agent.
- The environment contains a number of hidden tornados, represented by the character 't' in certain cells.
- The environment provides feedback to the agent after each probe or flag action, indicating the number of adjacent tornados in the probed cell.

Actuators:
- The agent can probe a cell by calling the probe(x,y) method, where x and y are the coordinates of the cell to be probed.
- The agent can flag a cell by calling the setFlag({x,y}) method, where x and y are the coordinates of the cell to be flagged.

Sensors:
The agent can observe the state of the game board by accessing the cellState variable of the Game object.

2.2 System Overview

The design of the system involves a Game class representing the game board, with Cell objects representing each cell on the board. The Agent class is the parent class for the BasicAgent, Beginner Agent and IntermediateAgent classes, which implement different strategies for playing the game. The system uses various methods and variables to interact with the game board environment and provide feedback to the agent.

2.2.1 Intialising the Game Board

The Game class is responsible for setting up the game board and managing the state of the game. Each of its cells can either contain a clue of the form 0-6 or a tornado. To initialise the game board, the Game class takes a 2D char array as input, which represents the world provided by the user. The Game class first determines the size of the board based on the length of the char array. It then scans through the char array to keep track of the number of tornadoes which can later be used to determine the outcome of the game, such as, whether the agent has managed to find all the tornados and wins or does not and dies.

Initially, the Game class sets the gameState array with '?' for each cell, representing that the state of each cell is unknown at the start of the game.

Once the game board has been initialised, the Game class can be used to start the game and play it by probing cells and updating the state of the board.

2.2.2 Basic Agent

The basic agent is a very naive way of solving the tornado sweeper problem with not much success. This is initialised by instantiating the `BasicAgent()`Class. This inherits the Agent Class which has all the common functionalities for the different agents. It starts off by first probing the safe cells. Based on the information received after probing the cell, it could either use the `recursivelyProbeZeros()` function to probe all the neighbours of the cell with a ‘0’. Else if there are no ‘0’s it probes the first top left cell that is not probed. The Basic agent's strategy of picking the top left unprobed cell is not ideal as this decision is made without taking into consideration any of the clues surrounding it. Hence, the agent dies almost every time and is not very successful at playing the game.

2.2.3 Beginner Agent

The BeginnerAgent class uses a simple strategy known as the Single Point Strategy (SPS). This strategy is based on identifying cells that have a single safe neighbouring cell, where the safe neighbouring cell has a clue equal to the number of uncovered cells in the rest of the neighbouring cells. The basic idea is to flag the covered cells as mines and probe the safe cell.

The implementation of this strategy can be described as follows:

1. Initialize an empty list to store all unprobed cells.
2. Probe the initial cell and middle cell.
3. If they have a clue of 0, recursively probe all safe cells.
4. While there are unmarked (unprobed and unflagged cells).
    a. For each neighbouring cell that has been probed, count the number of neighbouring cells that are flagged, and the number of neighbouring cells that are unmarked.
    b. If the number of flags is equal to the clue value, probe the unprobed cell.
    c. If the difference between the clue value and the number of flags is equal to the number of unprobed and unflagged cells, flag the unmarked cell.
    d. If the cell is neither probed nor flagged, add it to the list of unmarked cells.

5. Repeat steps 4-5 until there are no more unprobed cells or the maximum number of steps is reached.

2.2.4 Intermediate Agent

The intermediate agent uses a combination of the Single Point Strategy (SPS) and the Satisfiability test reasoning strategy (SATS) to determine whether a cell is safe to probe or should be examined further.

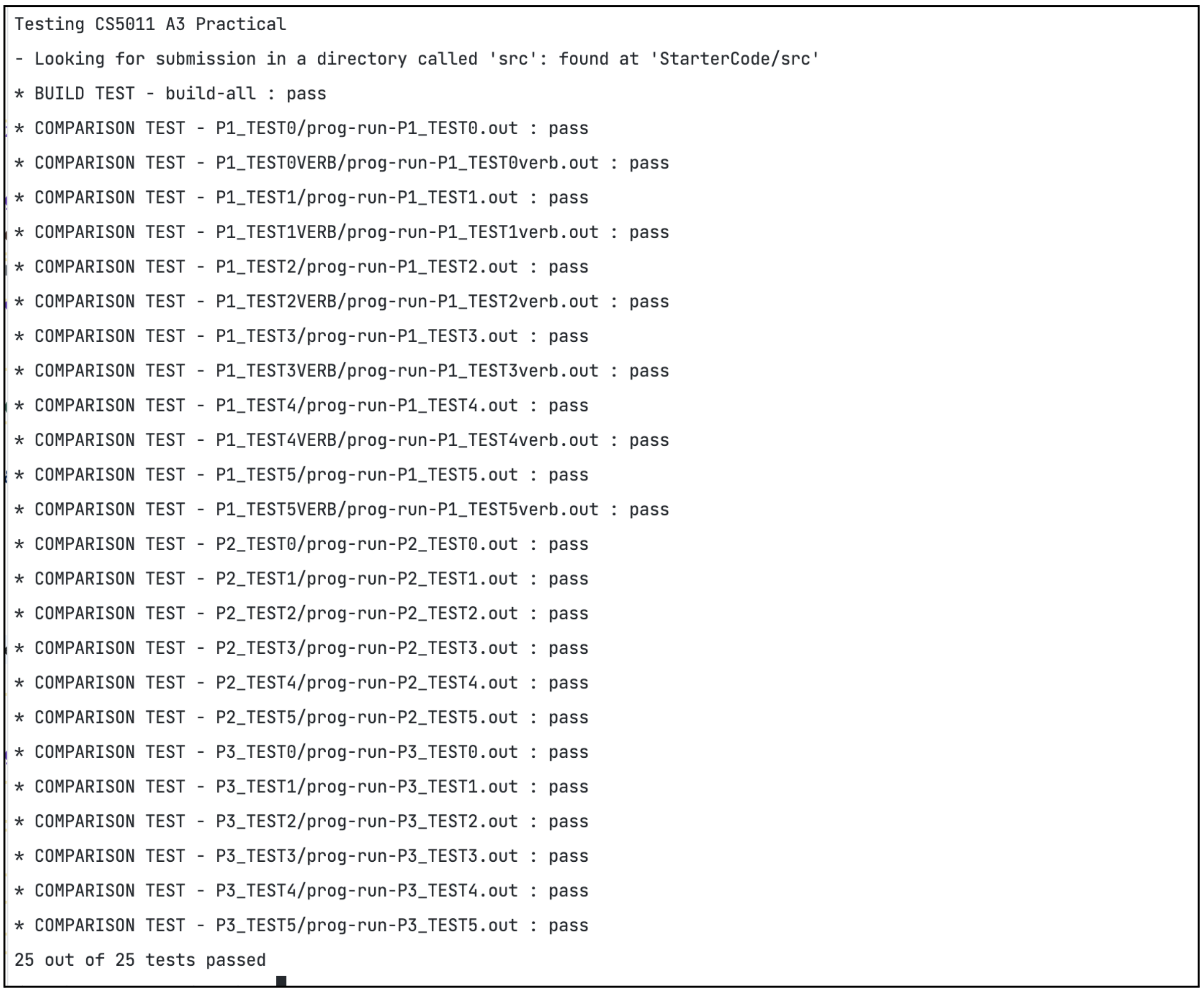
1. The agent first probes the two safe cells, (0,0) and the center cell, and recursively probes the adjacent neighbors of any 0 cells.
2. It then creates a knowledge base by constructing a string representation of all the probed cells.
3. Using this Knowledge Base String and logically ANDing it with the undiscovered cell, the agent uses the LogicNG's SAT solver to determine if this undiscovered cell is safe to probe. If the solver returns that the cell is safe to probe, the agent proceeds to probe the cell (Note that on every probe the Knowledge base string gets updated for any futuristic decisions). If the solver returns that the cell is not safe to probe, the agent uses the SPS strategy. The SPS strategy works save as discussed in the Beginner Agent's strategy.
4. The agent keeps repeating this untill there are no more undiscovered cells left.

The agent creates clauses that represent all possible combinations of safe and dangerous cells around the selected cell. The agent then uses the SAT solver to check whether any of the clauses are satisfiable. If a clause is satisfiable, the agent probes the selected cell. If none of the clauses are satisfiable, the agent flags the selected cell as dangerous and continues the search.

In summary, the intermediate agent uses the SPS strategy in combination with the SAT solver to determine the safety of an undiscovered cell. If the SAT solver returns that the cell is safe, the agent proceeds to probe the cell. If the SAT solver returns that the cell is not safe, the agent uses the SPS strategy to select a cell that has the highest probability of being safe to probe. The agent then uses the SAT solver to check the safety of the selected cell.

3. TESTING

Testing the program on the prebuilt test cases (standard testing with stacscheck):

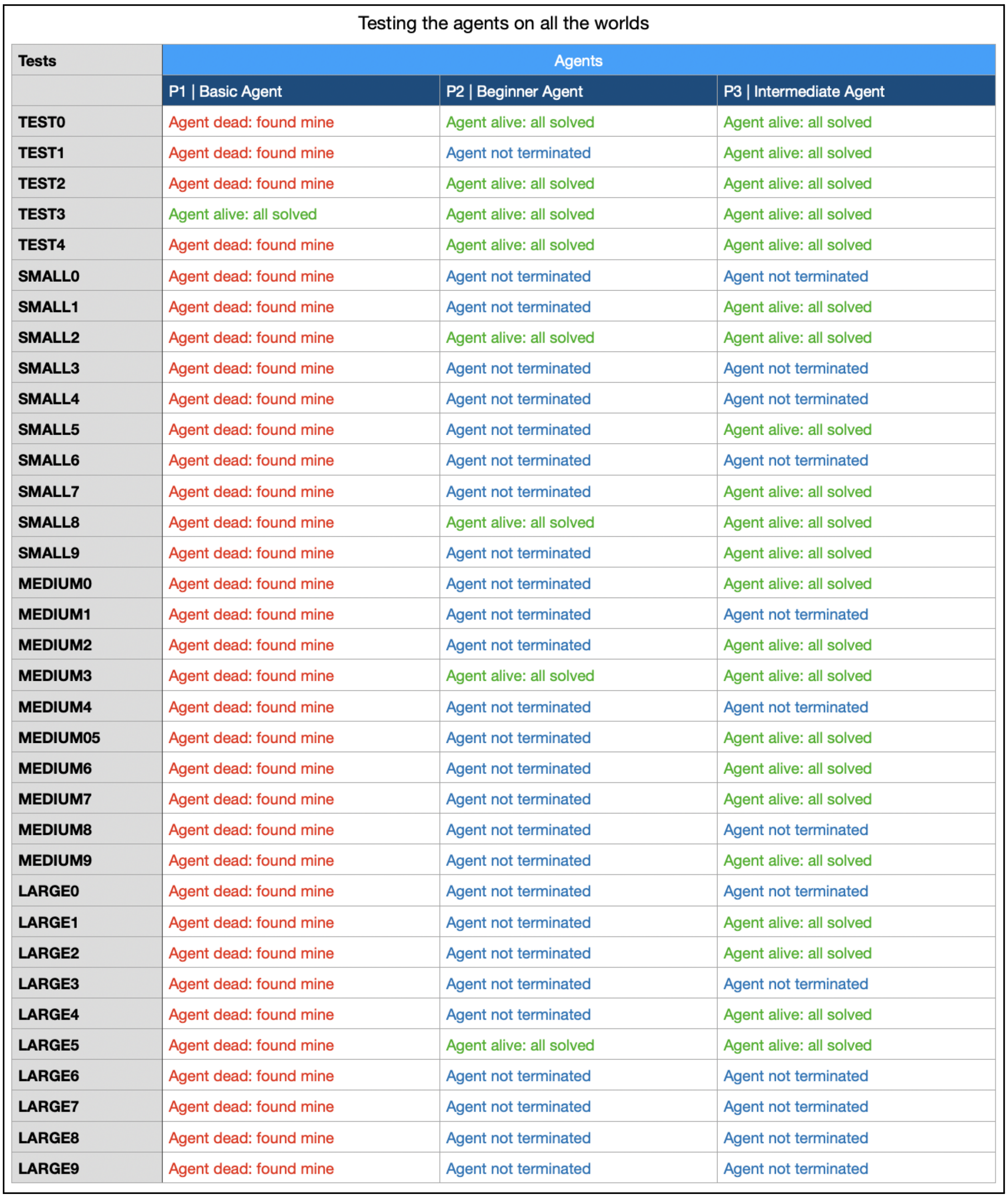


```
Testing CS5011 A3 Practical
- Looking for submission in a directory called 'src': found at 'StarterCode/src'
* BUILD TEST - build-all : pass
* COMPARISON TEST - P1_TEST0/prog-run-P1_TEST0.out : pass
* COMPARISON TEST - P1_TEST0VERB/prog-run-P1_TEST0verb.out : pass
* COMPARISON TEST - P1_TEST1/prog-run-P1_TEST1.out : pass
* COMPARISON TEST - P1_TEST1VERB/prog-run-P1_TEST1verb.out : pass
* COMPARISON TEST - P1_TEST2/prog-run-P1_TEST2.out : pass
* COMPARISON TEST - P1_TEST2VERB/prog-run-P1_TEST2verb.out : pass
* COMPARISON TEST - P1_TEST3/prog-run-P1_TEST3.out : pass
* COMPARISON TEST - P1_TEST3VERB/prog-run-P1_TEST3verb.out : pass
* COMPARISON TEST - P1_TEST4/prog-run-P1_TEST4.out : pass
* COMPARISON TEST - P1_TEST4VERB/prog-run-P1_TEST4verb.out : pass
* COMPARISON TEST - P1_TEST5/prog-run-P1_TEST5.out : pass
* COMPARISON TEST - P1_TEST5VERB/prog-run-P1_TEST5verb.out : pass
* COMPARISON TEST - P2_TEST0/prog-run-P2_TEST0.out : pass
* COMPARISON TEST - P2_TEST1/prog-run-P2_TEST1.out : pass
* COMPARISON TEST - P2_TEST2/prog-run-P2_TEST2.out : pass
* COMPARISON TEST - P2_TEST3/prog-run-P2_TEST3.out : pass
* COMPARISON TEST - P2_TEST4/prog-run-P2_TEST4.out : pass
* COMPARISON TEST - P2_TEST5/prog-run-P2_TEST5.out : pass
* COMPARISON TEST - P3_TEST0/prog-run-P3_TEST0.out : pass
* COMPARISON TEST - P3_TEST1/prog-run-P3_TEST1.out : pass
* COMPARISON TEST - P3_TEST2/prog-run-P3_TEST2.out : pass
* COMPARISON TEST - P3_TEST3/prog-run-P3_TEST3.out : pass
* COMPARISON TEST - P3_TEST4/prog-run-P3_TEST4.out : pass
* COMPARISON TEST - P3_TEST5/prog-run-P3_TEST5.out : pass
25 out of 25 tests passed
```

(figure 2: Stacscheck outcome)

To extend my testing and analysing the agents performance on the several other worlds, I've created a `test.sh` script to run over the different worlds for all the agents. I've also created a table listing the outcomes of the agents performance on the various worlds. (refer figure 3)

Testing the Program on all the worlds



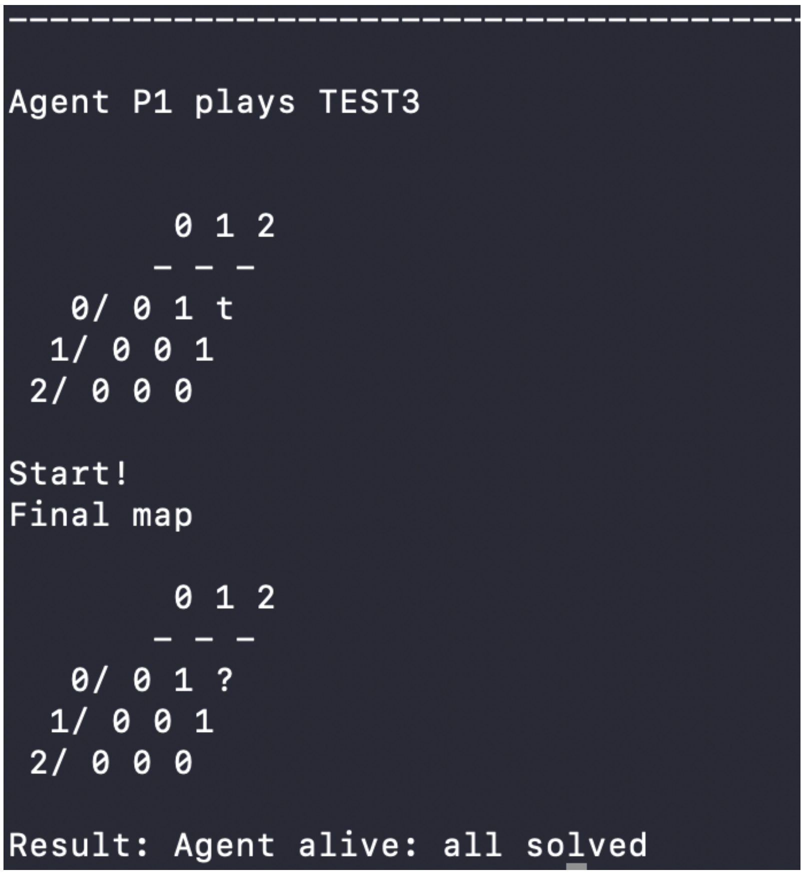
Testing the agents on all the worlds

Tests	Agents		
	P1 | Basic Agent	P2 | Beginner Agent	P3 | Intermediate Agent
TEST0	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
TEST1	Agent dead: found mine	Agent not terminated	Agent alive: all solved
TEST2	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
TEST3	Agent alive: all solved	Agent alive: all solved	Agent alive: all solved
TEST4	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
SMALL0	Agent dead: found mine	Agent not terminated	Agent not terminated
SMALL1	Agent dead: found mine	Agent not terminated	Agent alive: all solved
SMALL2	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
SMALL3	Agent dead: found mine	Agent not terminated	Agent not terminated
SMALL4	Agent dead: found mine	Agent not terminated	Agent not terminated
SMALL5	Agent dead: found mine	Agent not terminated	Agent alive: all solved
SMALL6	Agent dead: found mine	Agent not terminated	Agent not terminated
SMALL7	Agent dead: found mine	Agent not terminated	Agent alive: all solved
SMALL8	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
SMALL9	Agent dead: found mine	Agent not terminated	Agent alive: all solved
MEDIUM0	Agent dead: found mine	Agent not terminated	Agent alive: all solved
MEDIUM1	Agent dead: found mine	Agent not terminated	Agent not terminated
MEDIUM2	Agent dead: found mine	Agent not terminated	Agent alive: all solved
MEDIUM3	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
MEDIUM4	Agent dead: found mine	Agent not terminated	Agent not terminated
MEDIUM05	Agent dead: found mine	Agent not terminated	Agent alive: all solved
MEDIUM6	Agent dead: found mine	Agent not terminated	Agent alive: all solved
MEDIUM7	Agent dead: found mine	Agent not terminated	Agent alive: all solved
MEDIUM8	Agent dead: found mine	Agent not terminated	Agent not terminated
MEDIUM9	Agent dead: found mine	Agent not terminated	Agent alive: all solved
LARGE0	Agent dead: found mine	Agent not terminated	Agent not terminated
LARGE1	Agent dead: found mine	Agent not terminated	Agent alive: all solved
LARGE2	Agent dead: found mine	Agent not terminated	Agent alive: all solved
LARGE3	Agent dead: found mine	Agent not terminated	Agent not terminated
LARGE4	Agent dead: found mine	Agent not terminated	Agent alive: all solved
LARGE5	Agent dead: found mine	Agent alive: all solved	Agent alive: all solved
LARGE6	Agent dead: found mine	Agent not terminated	Agent not terminated
LARGE7	Agent dead: found mine	Agent not terminated	Agent not terminated
LARGE8	Agent dead: found mine	Agent not terminated	Agent not terminated
LARGE9	Agent dead: found mine	Agent not terminated	Agent not terminated

(figure 3: Testing the program on several worlds and recording the outcomes of the agents performance)

Lets now examine a few of these test cases and see why each of the agent fails for a few edge cases.

P1 plays TEST3 and wins
If you look at figure 3 it shows how unsuccessful the the agent is. Lets analyse the agent playing the TEST3 in order to see how it won.

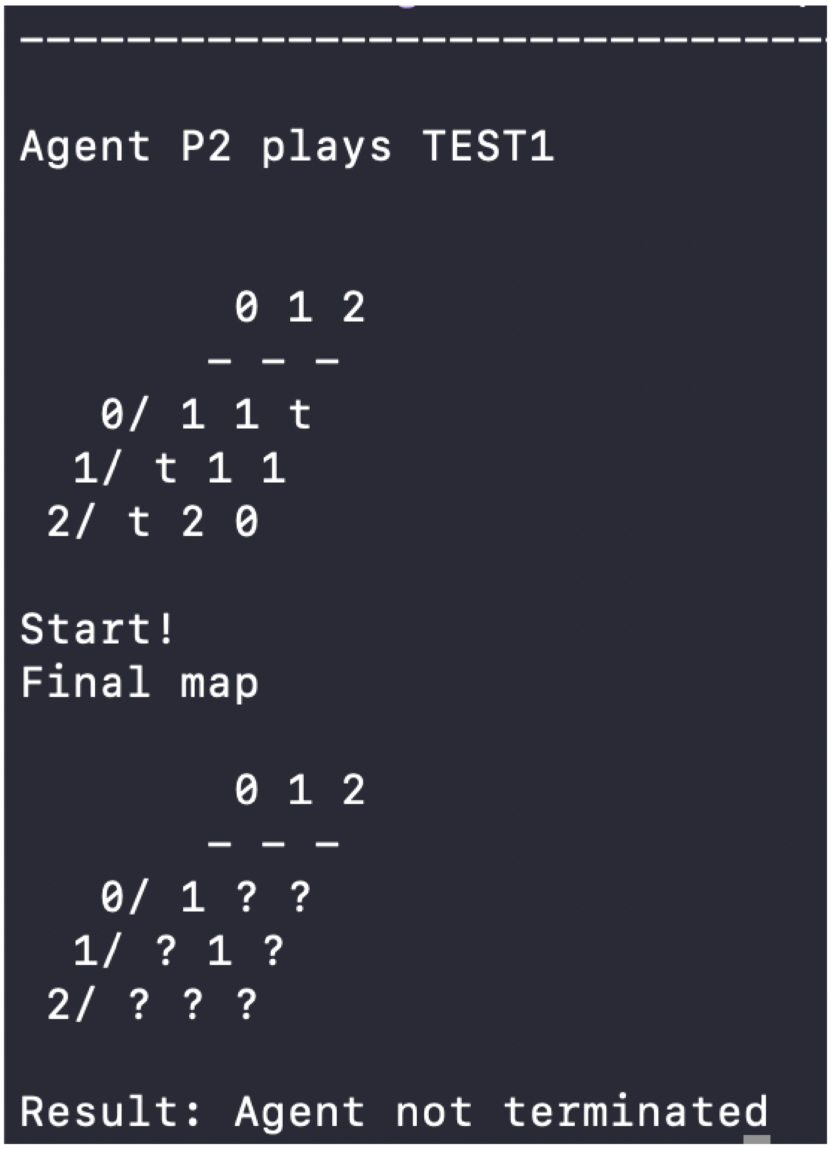


```
----------------------------------------

Agent P1 plays TEST3


        0 1 2
       - - -
   0/ 0 1 t
  1/ 0 0 1
 2/ 0 0 0

Start!
Final map

        0 1 2
       - - -
   0/ 0 1 ?
  1/ 0 0 1
 2/ 0 0 0

Result: Agent alive: all solved
```

(figure 4: Testing the Basic Agent on TEST3 to see how it wins)

The agent starts at (0, 0) and immediately encounters a ‘0’ and hence, recursively probes its neighbours as its safe. On doing so it probes more ‘0’s as the map is filled with it in the bottom. It stops when there are no more recursive probes and its just left with one cell in the map. Since it hasn’t died and thats the last cell it has to be the tornado and it flags it. This is an edge case which is very rare and hence the agent is only able to pass this case in spite of using a naive strategy.

P2 plays TEST1 and doesn’t terminate
If you look at figure 3 it shows how unsuccessful the the agent is. Lets analyse the agent playing the TEST3 in order to see how it won.

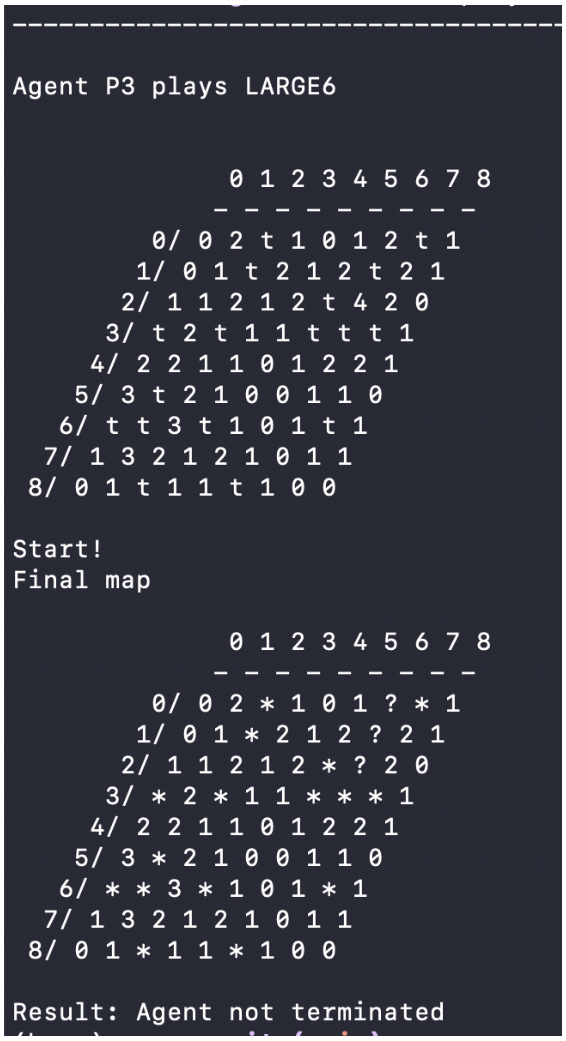


```
-----------------------------------

Agent P2 plays TEST1


       0 1 2
      - - -
   0/ 1 1 t
  1/ t 1 1
 2/ t 2 0

Start!
Final map

       0 1 2
      - - -
   0/ 1 ? ?
  1/ ? 1 ?
 2/ ? ? ?

Result: Agent not terminated
```

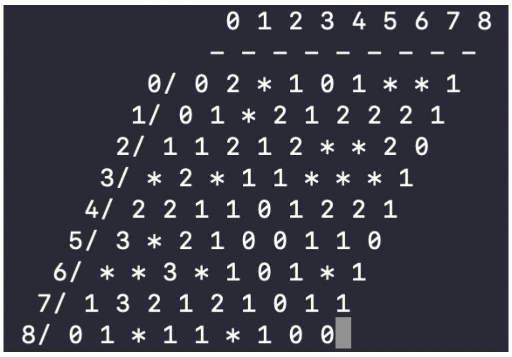
(*figure 5: Testing the Beginner Agent on TEST1 to see why it doesn't terminate)*

The Beginner agent starts by probing the safe cells (0,0) and (1,1), however, no recursive probes could be made leaving the agen with these cluse at the start. It then starts to apply the SP strategy to every undiscovered cell. The SP strategy works by counting checking two conditions: AFN and AMN. If the AFN condition for a cell suffices then the cell is safe to probe else if the AMN condition suffices then the cell is unsafe and must be flagged. However, if none of these conditions hold then its safe to come back to this cell later after more information is discovered. However, in this case every other cell seems to suffer from the save problem wher the information at hand from the neighbours is not sufficient to determine if the cell is safe or not. This leads to the agent moving to the next cell in search of more info and applying SPS but gets stuck in a loop. This can be overcome by taking into consideration the knowledge of all the cells using a knowledge base which is seen to be implemented in the Intermediate agent and hence it overcomes this test case where the Beginner agent fails to prove useful.

P3 plays LARGE6 and doesn't terminate



```
----------------------------------------

Agent P3 plays LARGE6


              0 1 2 3 4 5 6 7 8
             - - - - - - - - - -
         0/ 0 2 t 1 0 1 2 t 1
        1/ 0 1 t 2 1 2 t 2 1
       2/ 1 1 2 1 2 t 4 2 0
      3/ t 2 t 1 1 t t t 1
     4/ 2 2 1 1 0 1 2 2 1
    5/ 3 t 2 1 0 0 1 1 0
   6/ t t 3 t 1 0 1 t 1
  7/ 1 3 2 1 2 1 0 1 1
 8/ 0 1 t 1 1 t 1 0 0

Start!
Final map

              0 1 2 3 4 5 6 7 8
             - - - - - - - - - -
         0/ 0 2 * 1 0 1 ? * 1
        1/ 0 1 * 2 1 2 ? 2 1
       2/ 1 1 2 1 2 * ? 2 0
      3/ * 2 * 1 1 * * * 1
     4/ 2 2 1 1 0 1 2 2 1
    5/ 3 * 2 1 0 0 1 1 0
   6/ * * 3 * 1 0 1 * 1
  7/ 1 3 2 1 2 1 0 1 1
 8/ 0 1 * 1 1 * 1 0 0

Result: Agent not terminated
```



```
              0 1 2 3 4 5 6 7 8
             - - - - - - - - - -
         0/ 0 2 * 1 0 1 * * 1
        1/ 0 1 * 2 1 2 2 2 1
       2/ 1 1 2 1 2 * * 2 0
      3/ * 2 * 1 1 * * * 1
     4/ 2 2 1 1 0 1 2 2 1
    5/ 3 * 2 1 0 0 1 1 0
   6/ * * 3 * 1 0 1 * 1
  7/ 1 3 2 1 2 1 0 1 1
 8/ 0 1 * 1 1 * 1 0 0
```

(*figure 6: Testing the Intermediate Agent on LARGE6 to see why it doesn't terminate and alternate solution)*

This is a very interesting case as the last unknowns are prone to having multiple solutions. The figure on the right is an alternate solution to the one displayed in the game. Due to no concrete evidence on which cell to be probed due to the ambiguity in such a case the agent fails to terminate inspite of the knowledge of the world. Even as a human trying to solve this problem, on getting to the last stage of unknowns we'd not be able to determine the most ideal choice.

4. EVALUATION

This section compares the performance of three agents that use different strategies to probe cells in the Tornado Sweeper game: the Basic move agent which probes the first undiscovered cell, the Basic agent which incorporated the single point strategy, and the Intermediate agent using SAT in combination with SPS .

Agent	Performance in terms of #tests passed percentage
P1 Basic Agent	2.85%
P2 Beginner Agent	22.85%
P3 Intermediate Agent	62.85%

The results show that the Basic agent had the lowest win rate, with an average of 2.85%, the single point strategy agent had a higher win rate, with an average of 22.85%, the Intermediate agent with a combination os SPS ans SAT got to a 62.85% win rate.

In summary, the comparison of the three agents can be summarized as follows:

Basic Agent:
Pros: easy to implement, perform depends on a few edge cases and may pass them in certain situations.
Cons: does not use strategic reasoning, performance may vary widely.

Beginner Agent:
Pros: more intelligent than the basic agent, can perform better in situations where the outcome of an undiscovered cell can be determined only by looking at its neighbours.
Cons: does not use deductive reasoning, may fail in situations with no clear structure.

Intermediate Agent:
Pros: most intelligent among the three, can perform well in a wide range of situations.
Cons: most complex to implement out of the two as it implements both the SPS and SAT strategies, may be slower than other two agents.

5. BIBLIOGRAPHY

Darrell Whitley and Don Piele, "The Tornado Sweeper: A Hybrid Agent for Minesweeper," Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, July 2004.

Andreas Ohliger, "LogicNG: A Modern Propositional Logic Toolkit," Journal of Automated Reasoning, vol. 59, no. 1, pp. 1-29, Aug. 2017.

Andreas Ohliger, "LogicNG: A Propositional Logic Toolkit in Java," arXiv:1502.04601, Feb. 2015. [Online]. Available: https://arxiv.org/abs/1502.04601. [Accessed: Mar. 29, 2023].