Minesweeper

Teacher Manual

By Justin Chung, John Lu, Jian Li, Sui Ting Yeung

Directed by Abdullah Younis

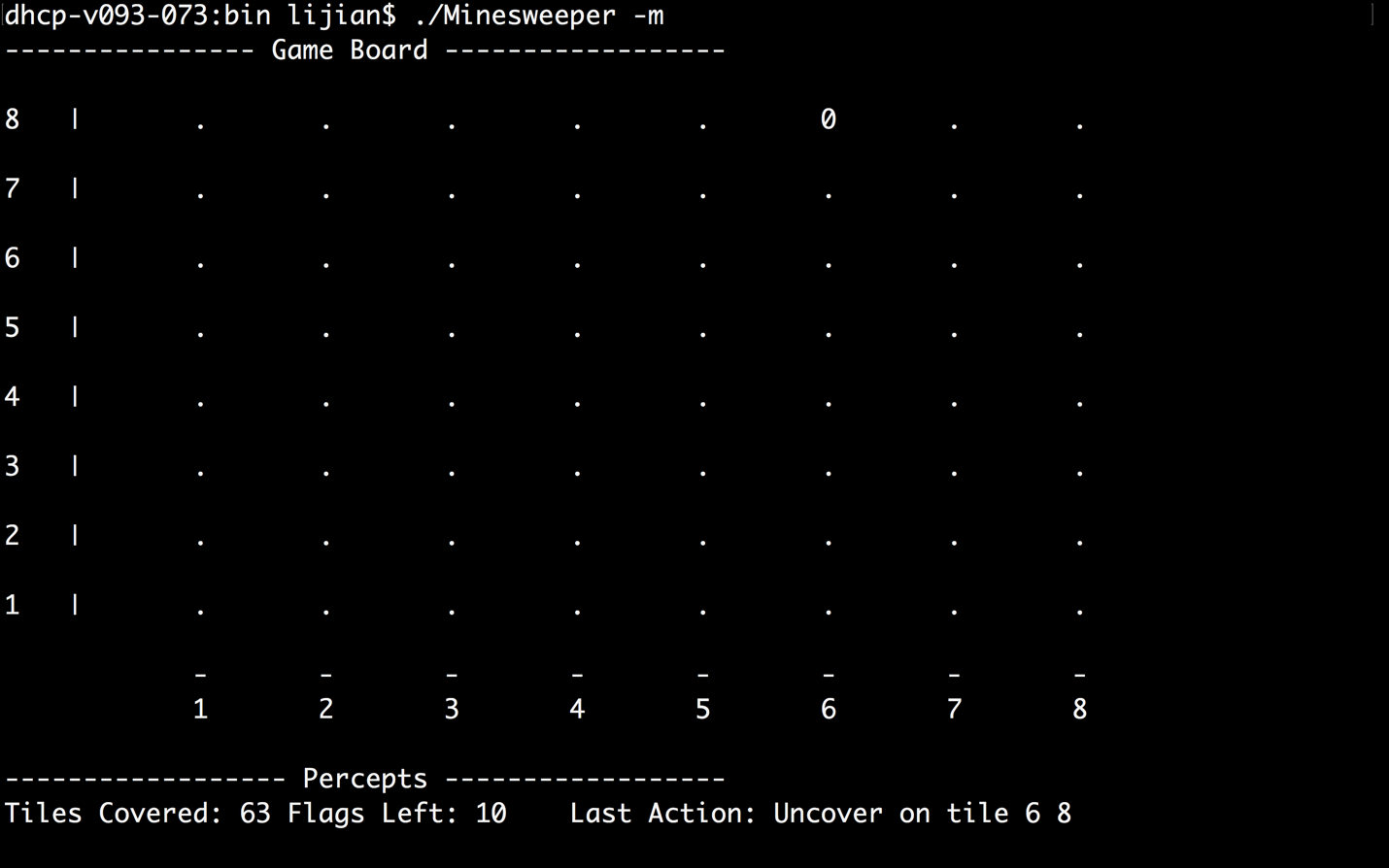
For CS-171

Table of contents

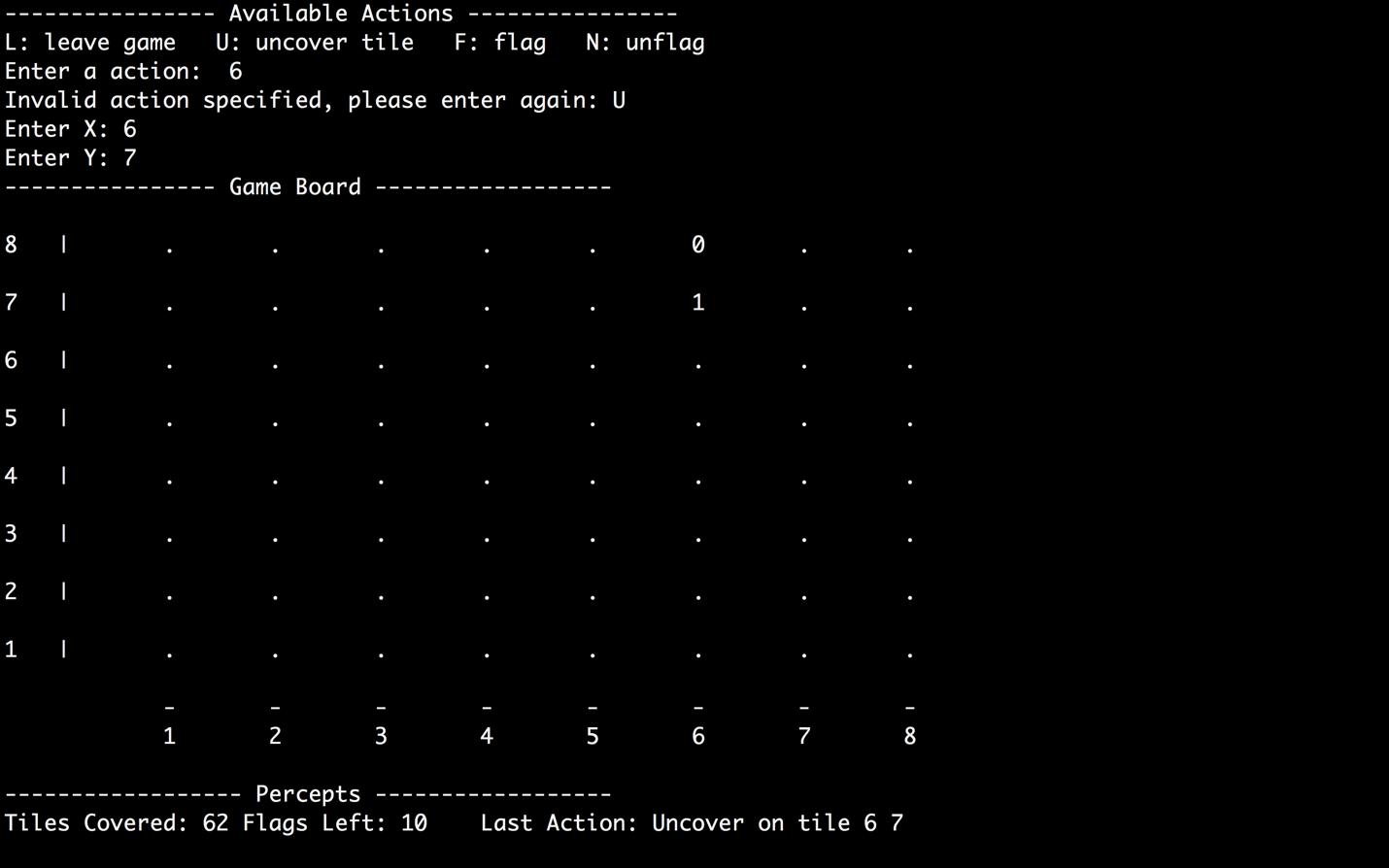
* User Case
* Method Explanation
* Scoring system and grading

**User Case**

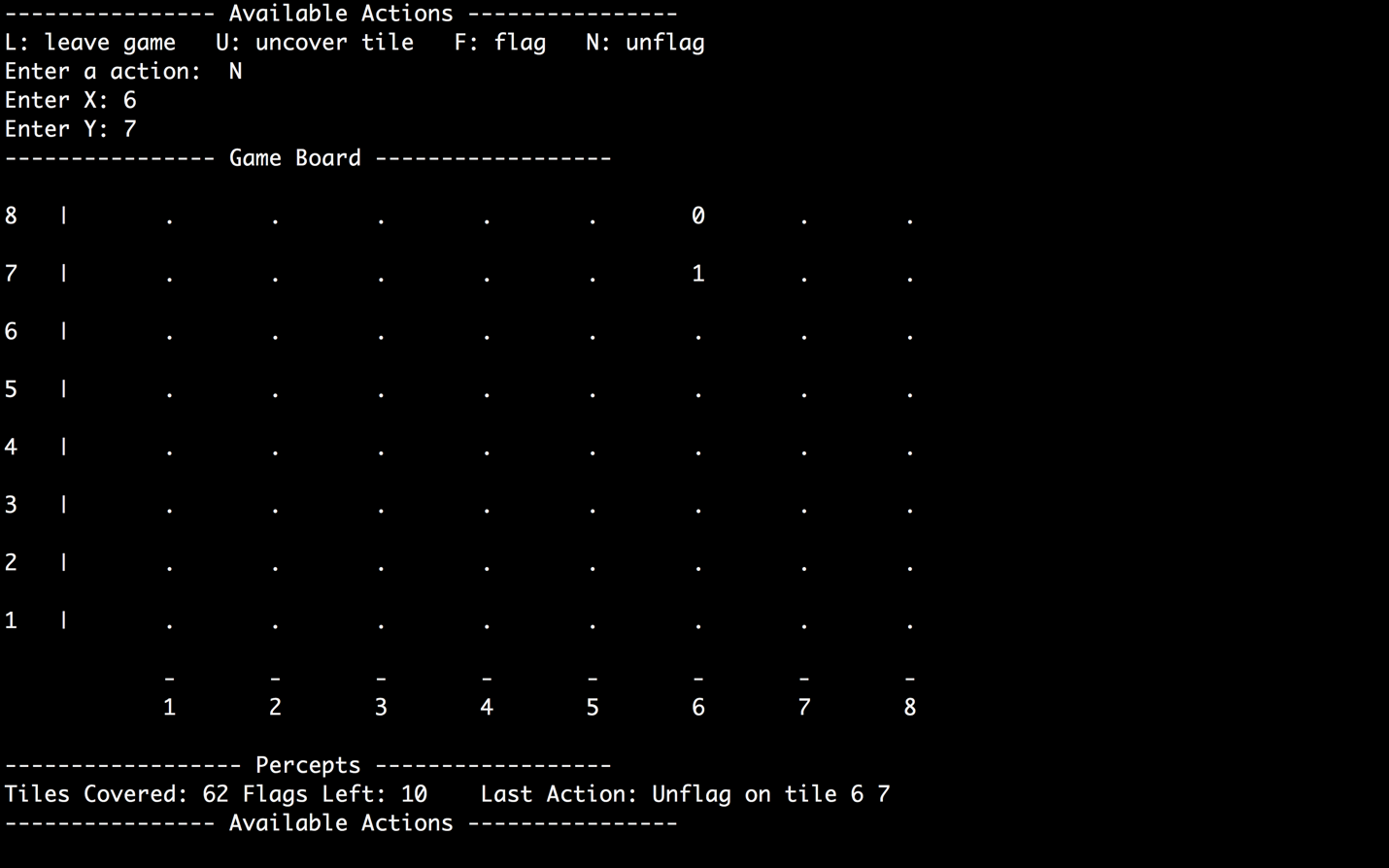
An example demonstrated by manual mode in beginner world.



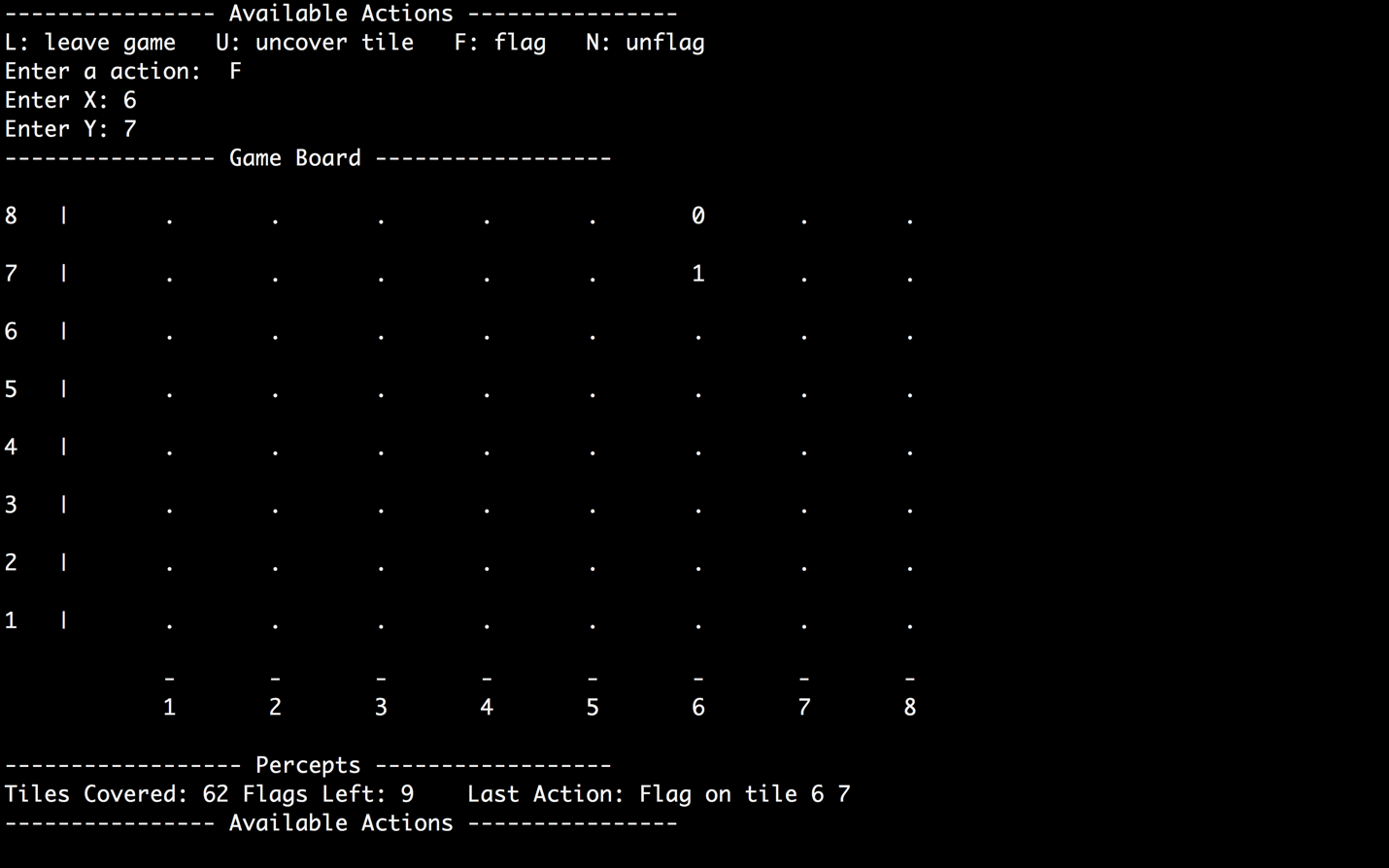
The shell uncover the first tile automatically, and the first move is not counted as user’s move.



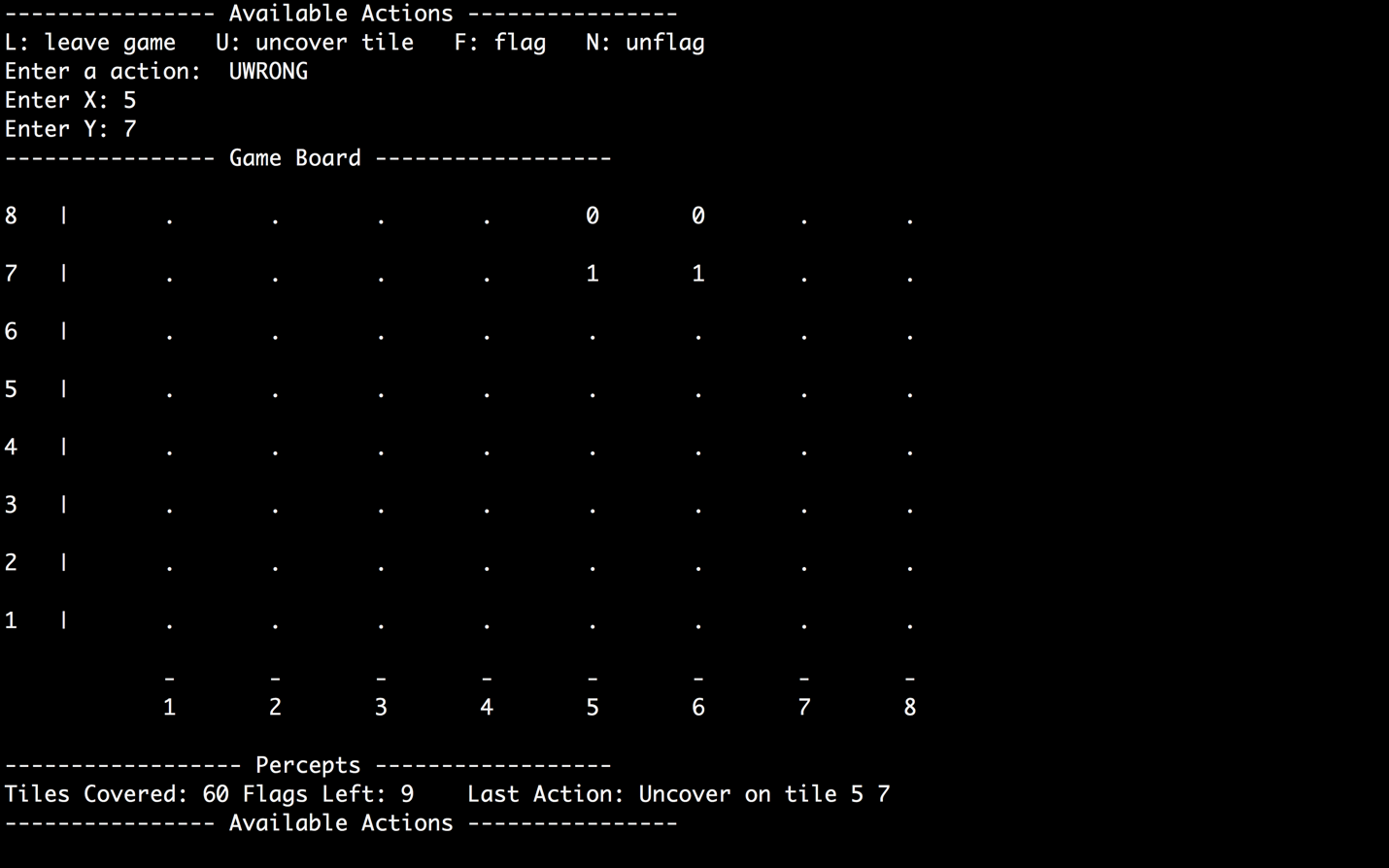
The shell discards 6 as a valid action\_type. Instead, valid user input is uncover 6,7. The actual raw input is Action{Action\_type::UNCOVER, 5, 6}



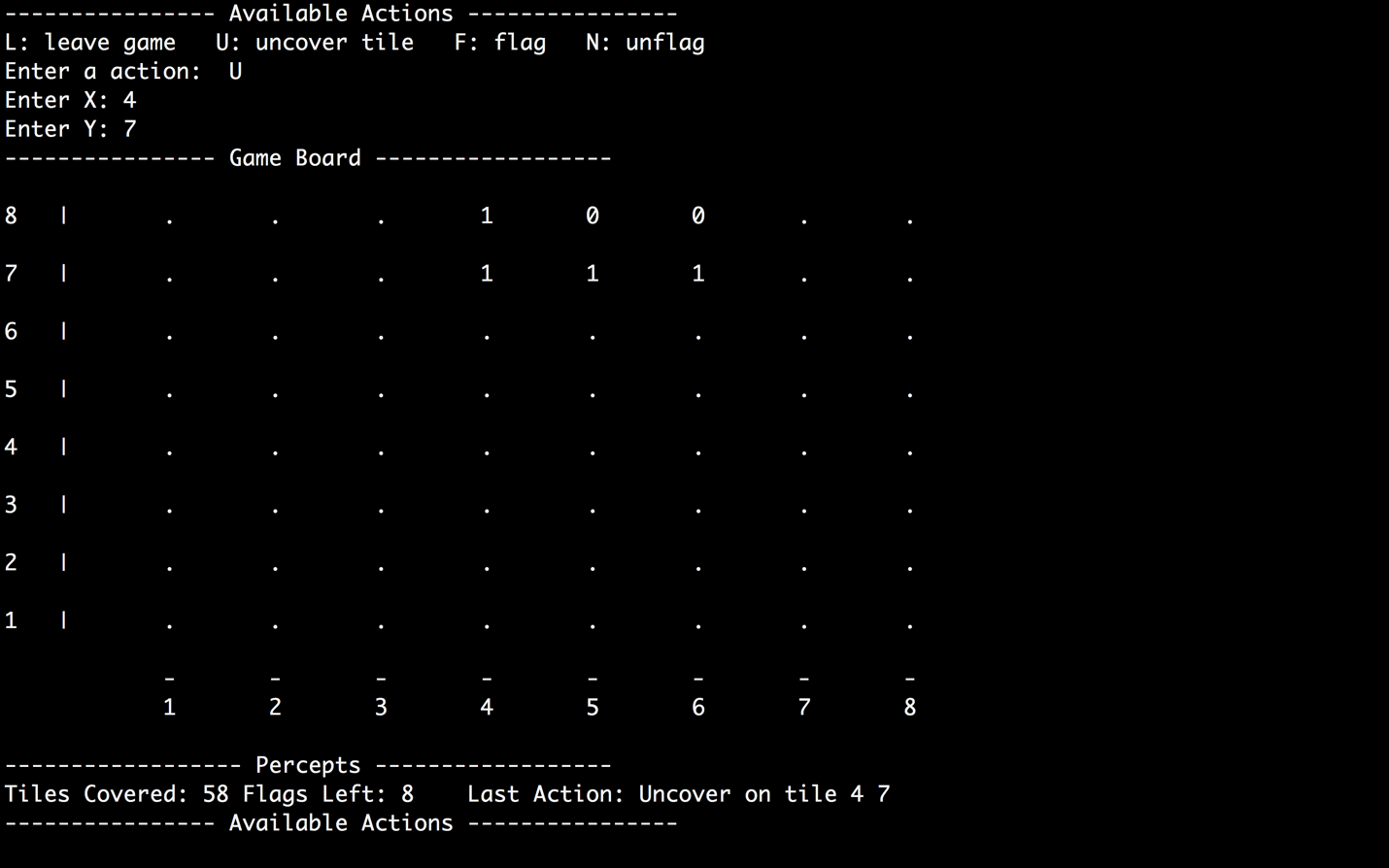
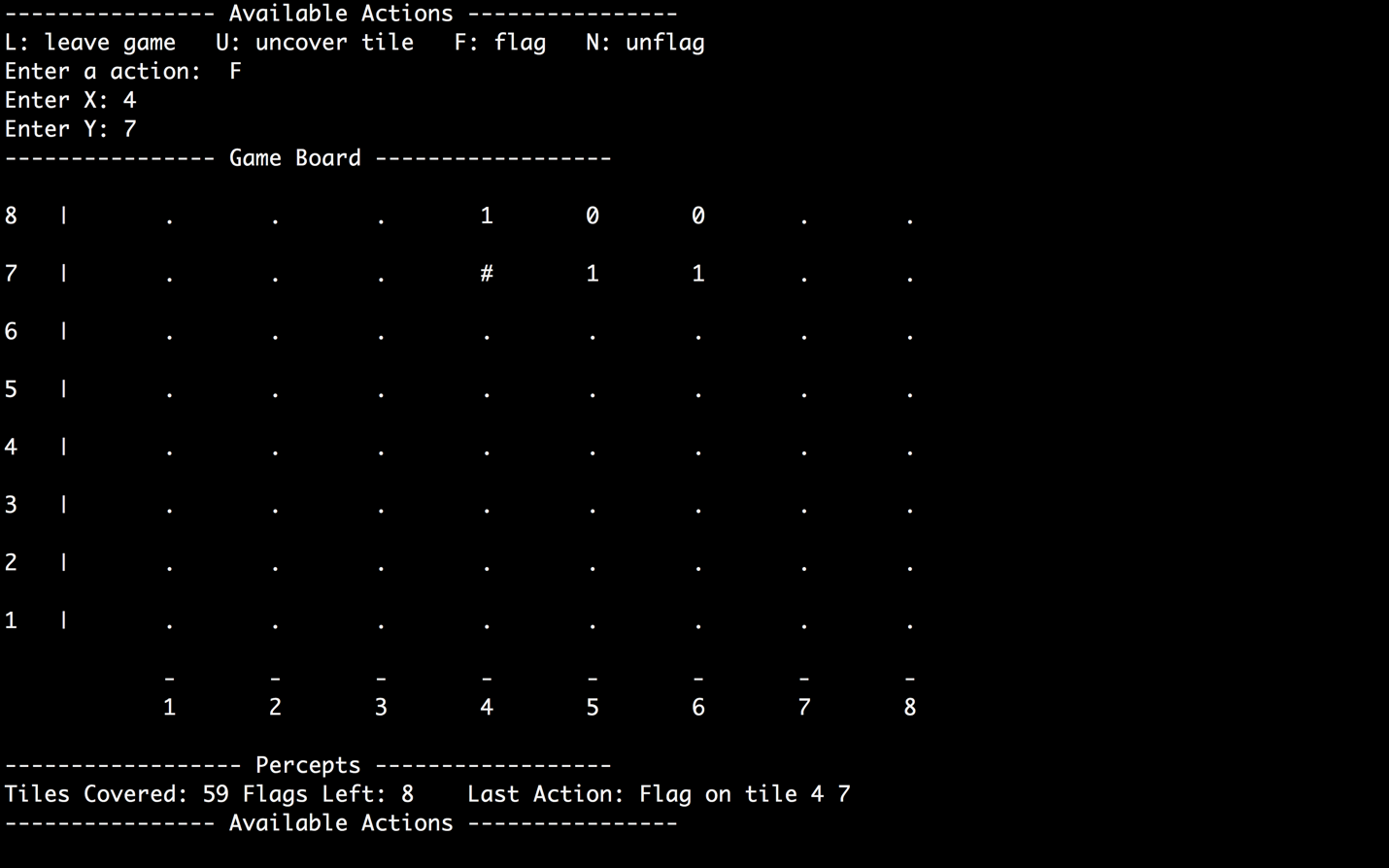
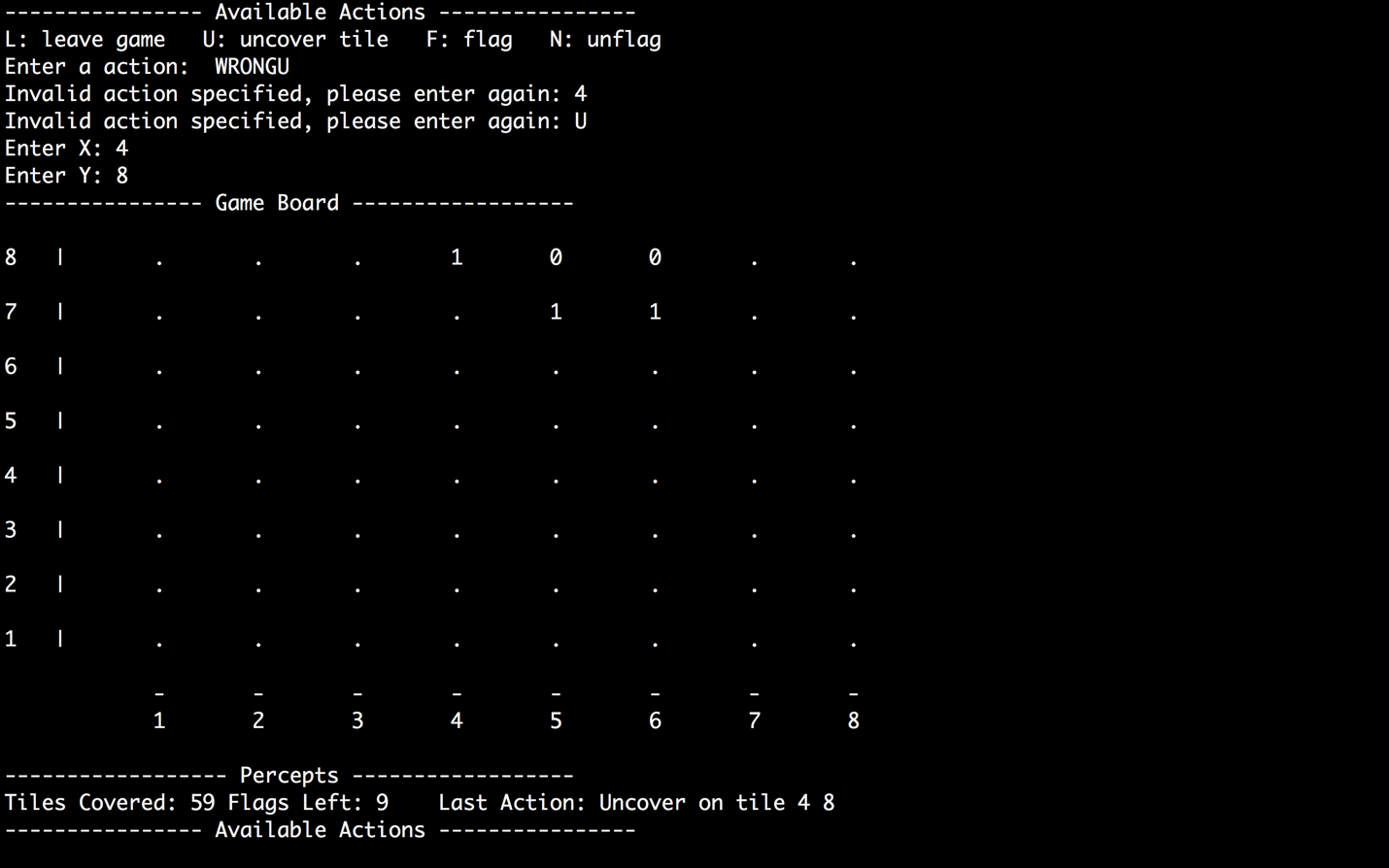
Un-flag a tile that is flagged before results in no effect on the board.



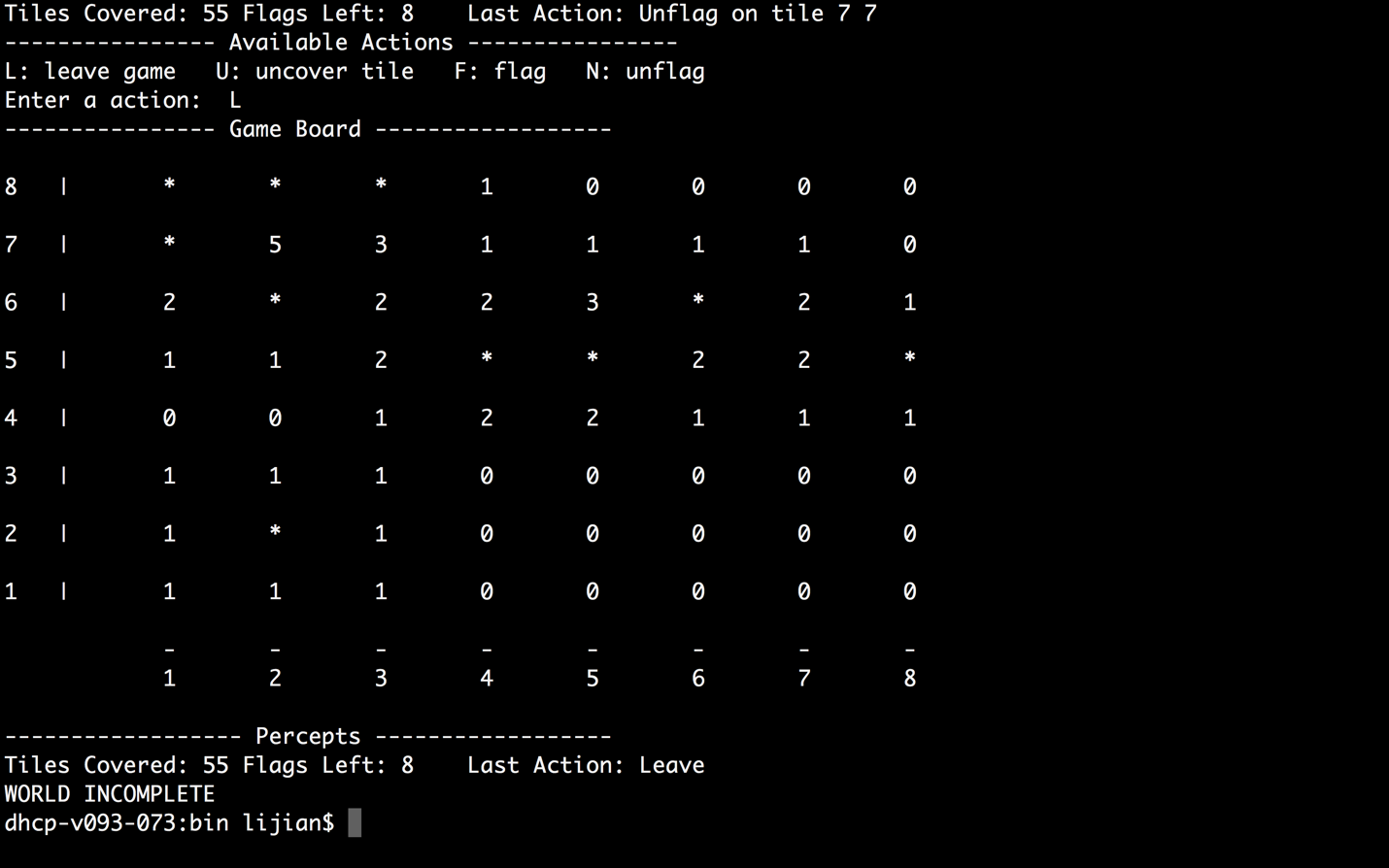
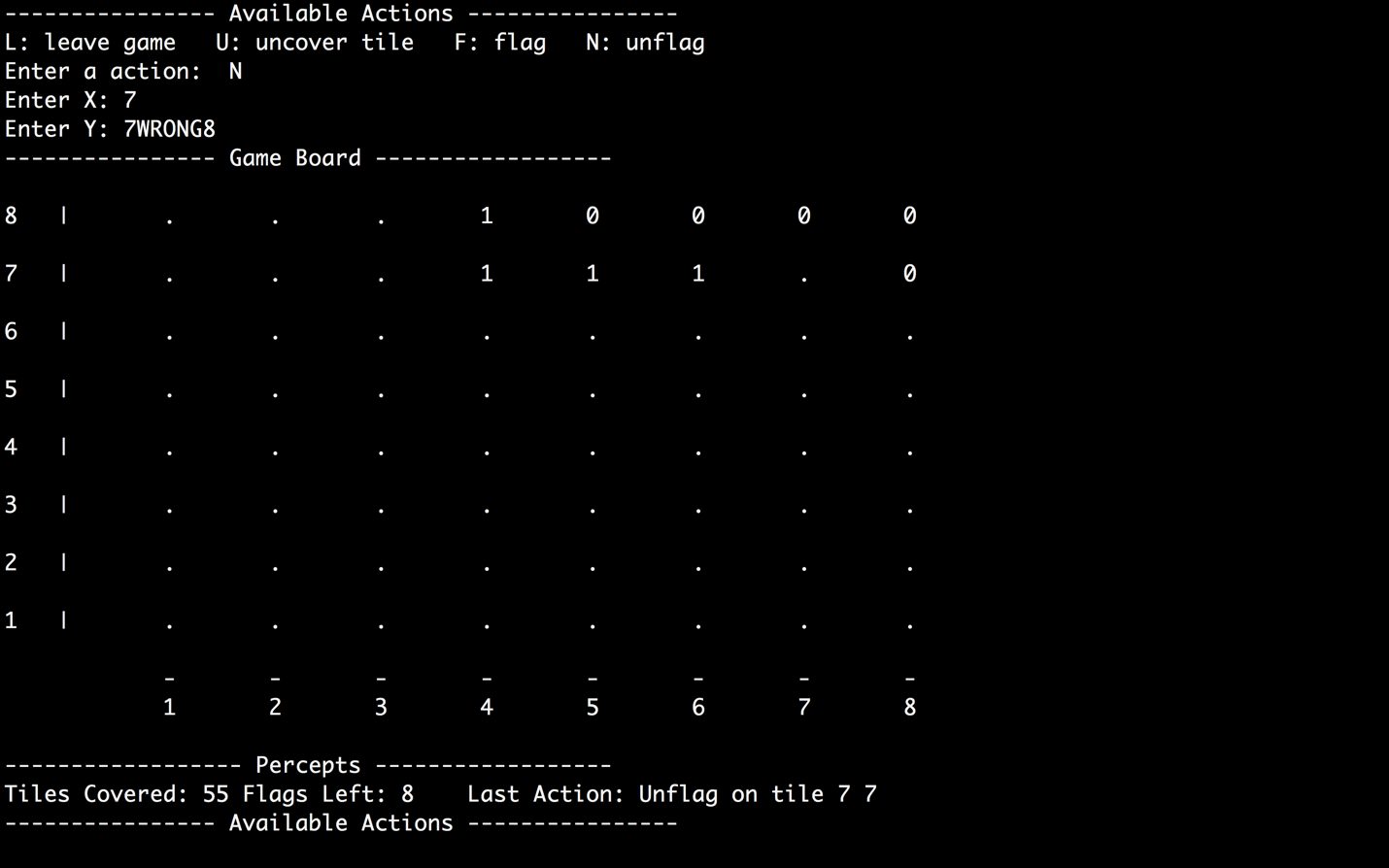
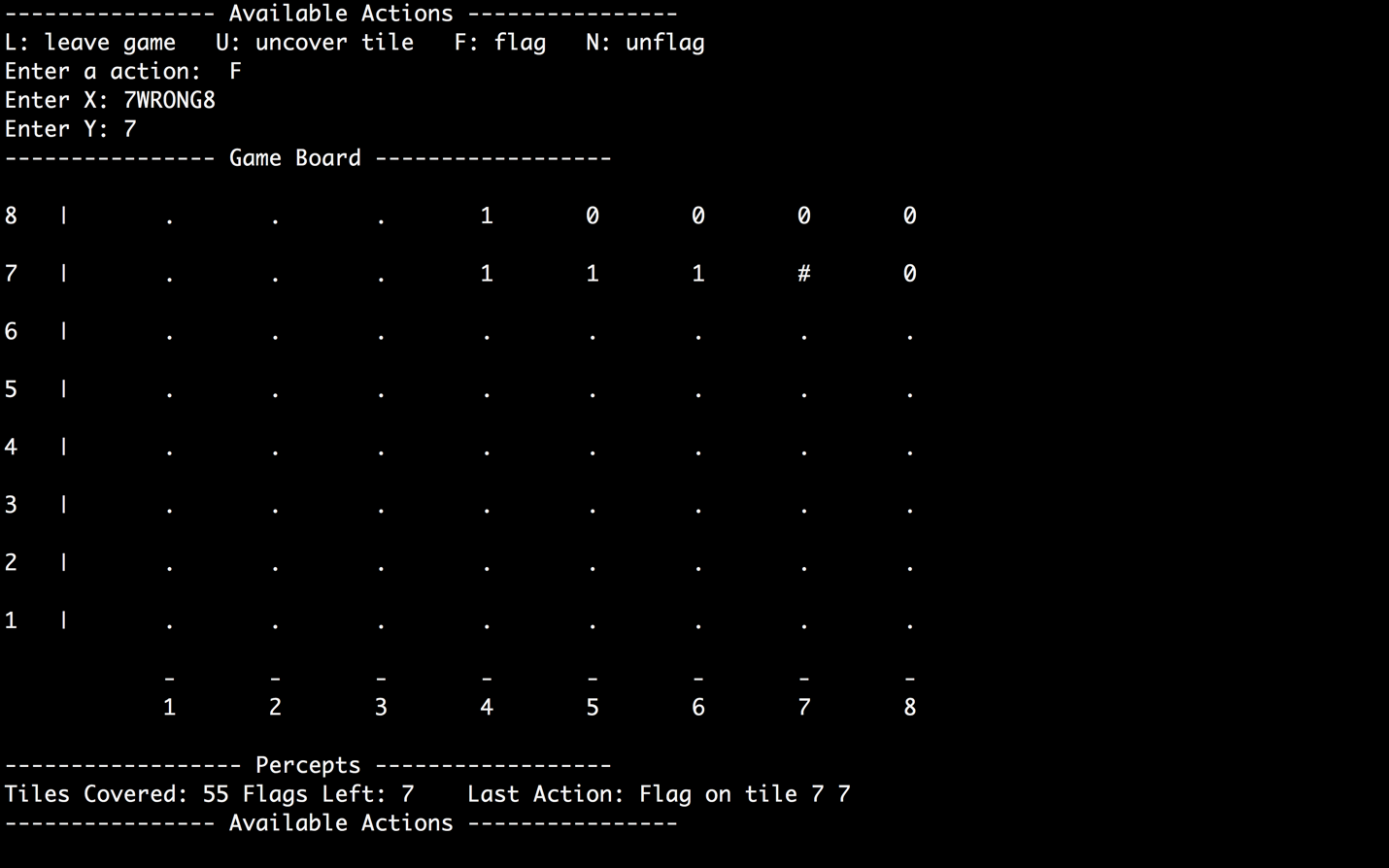
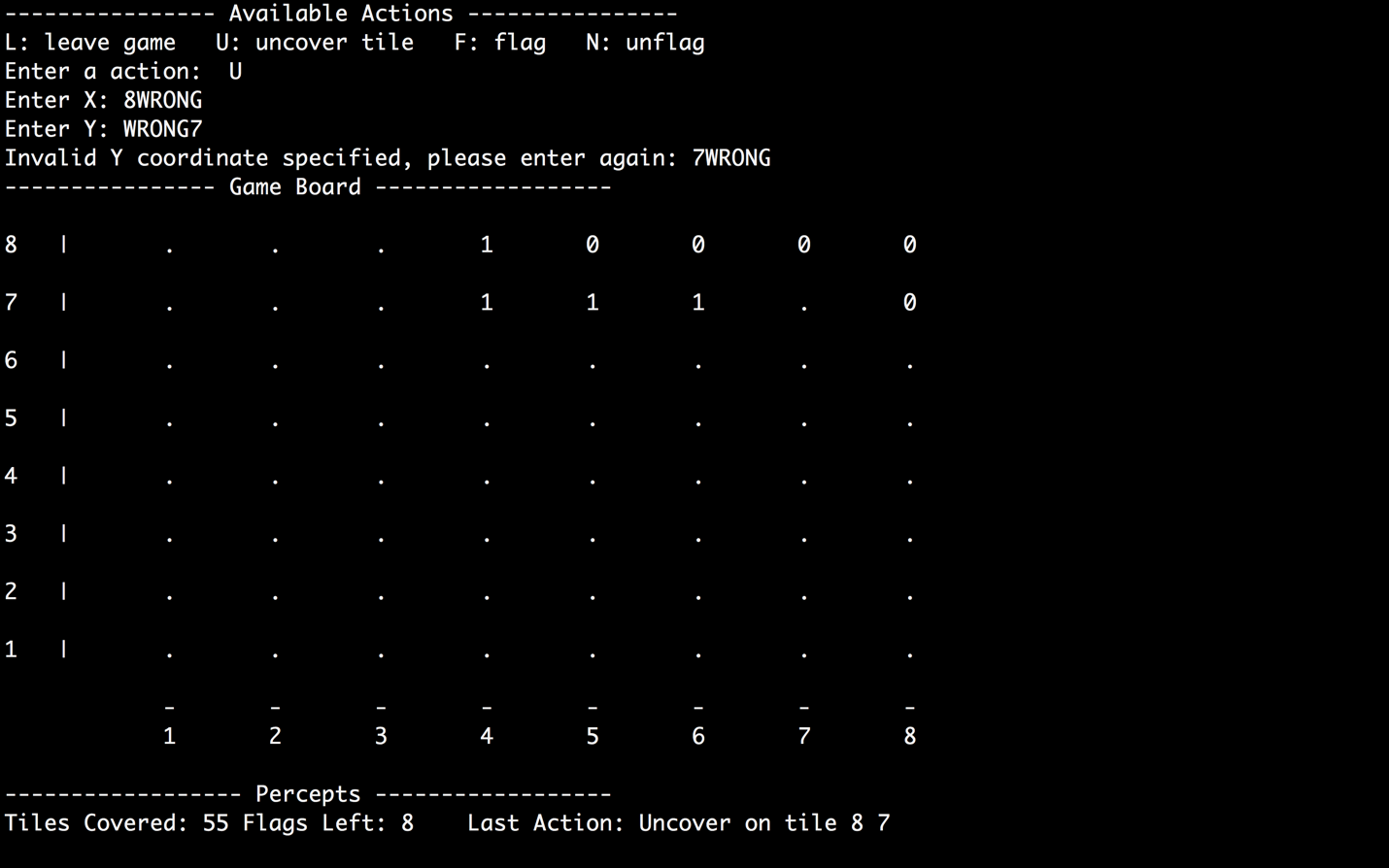
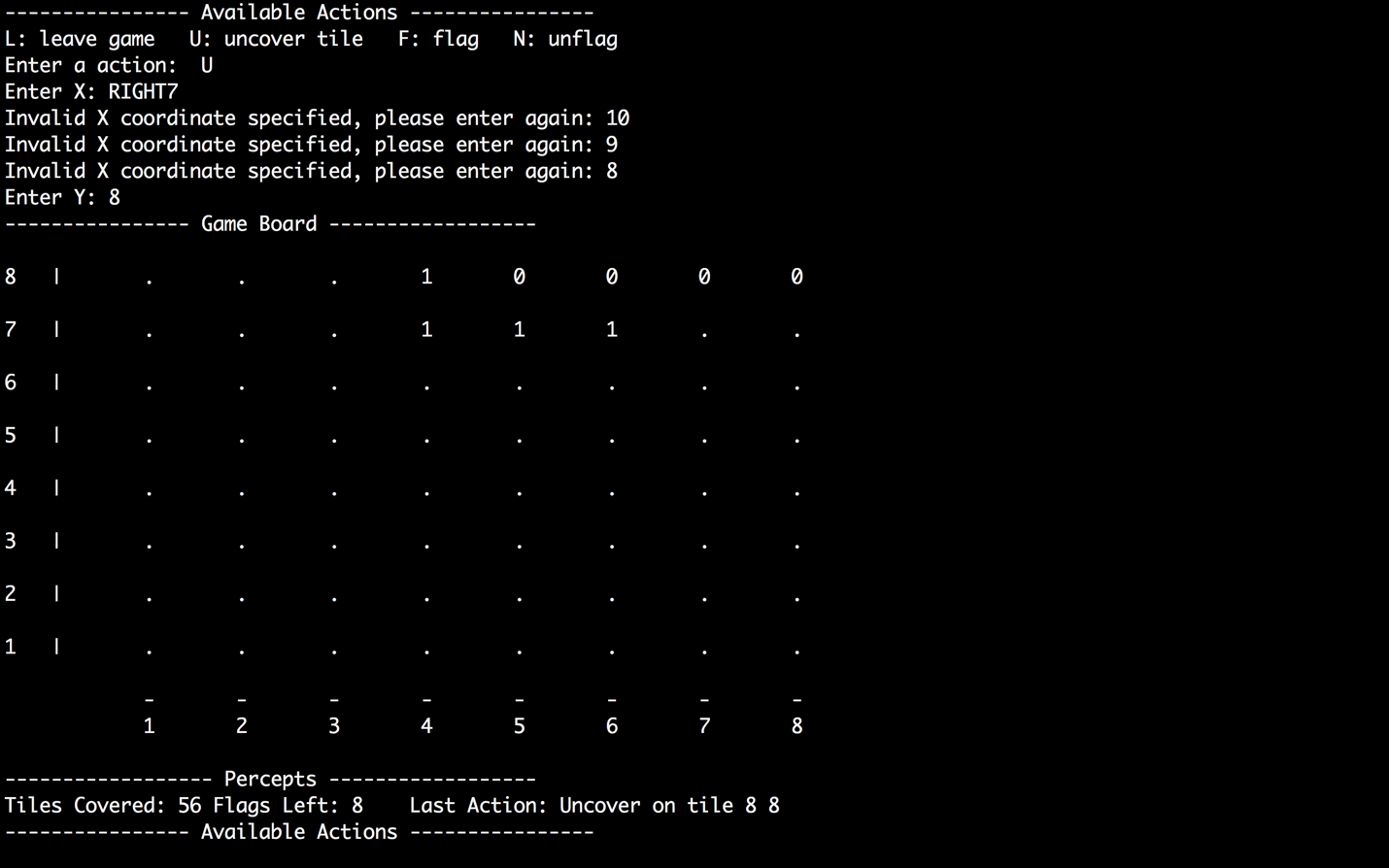
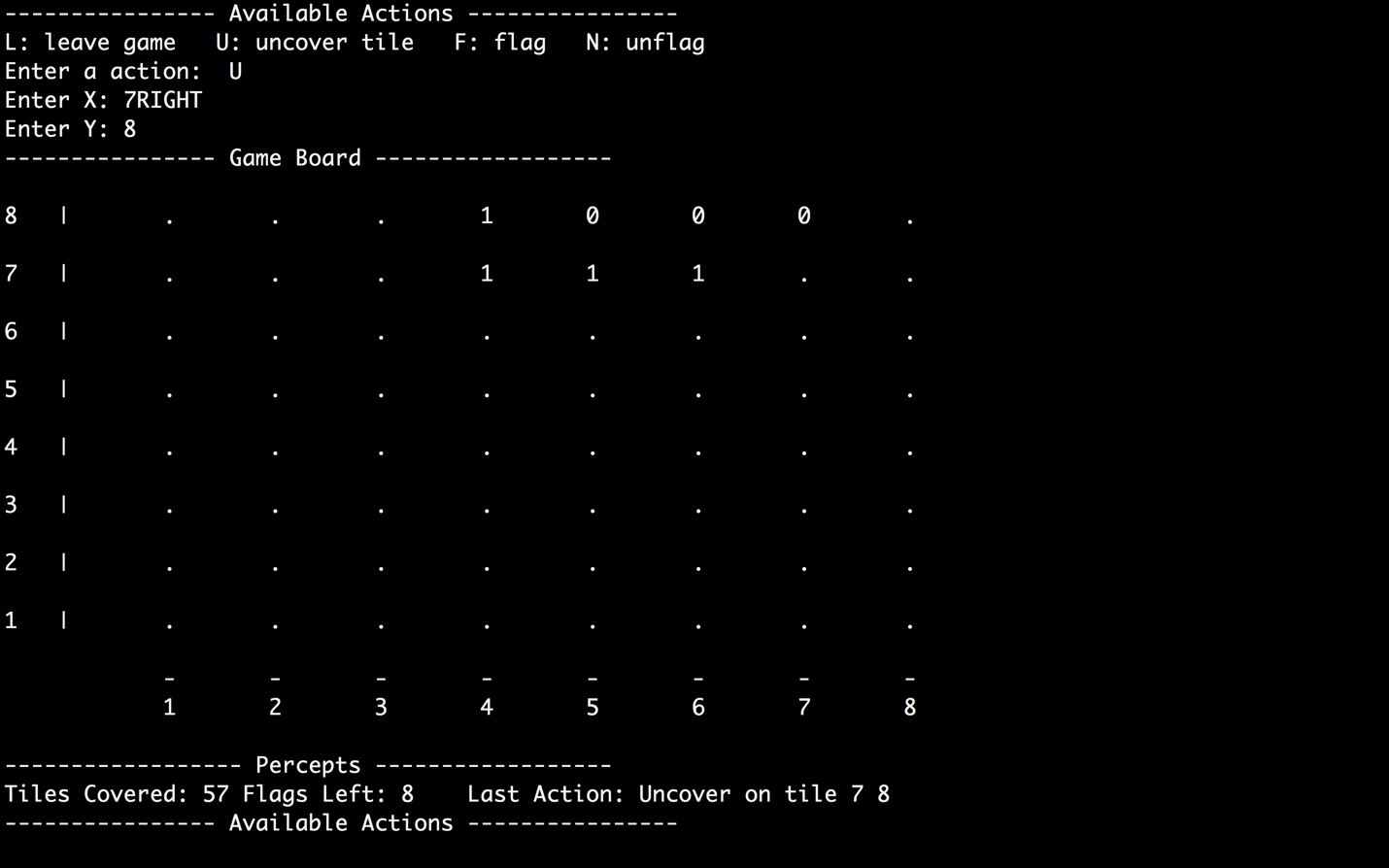
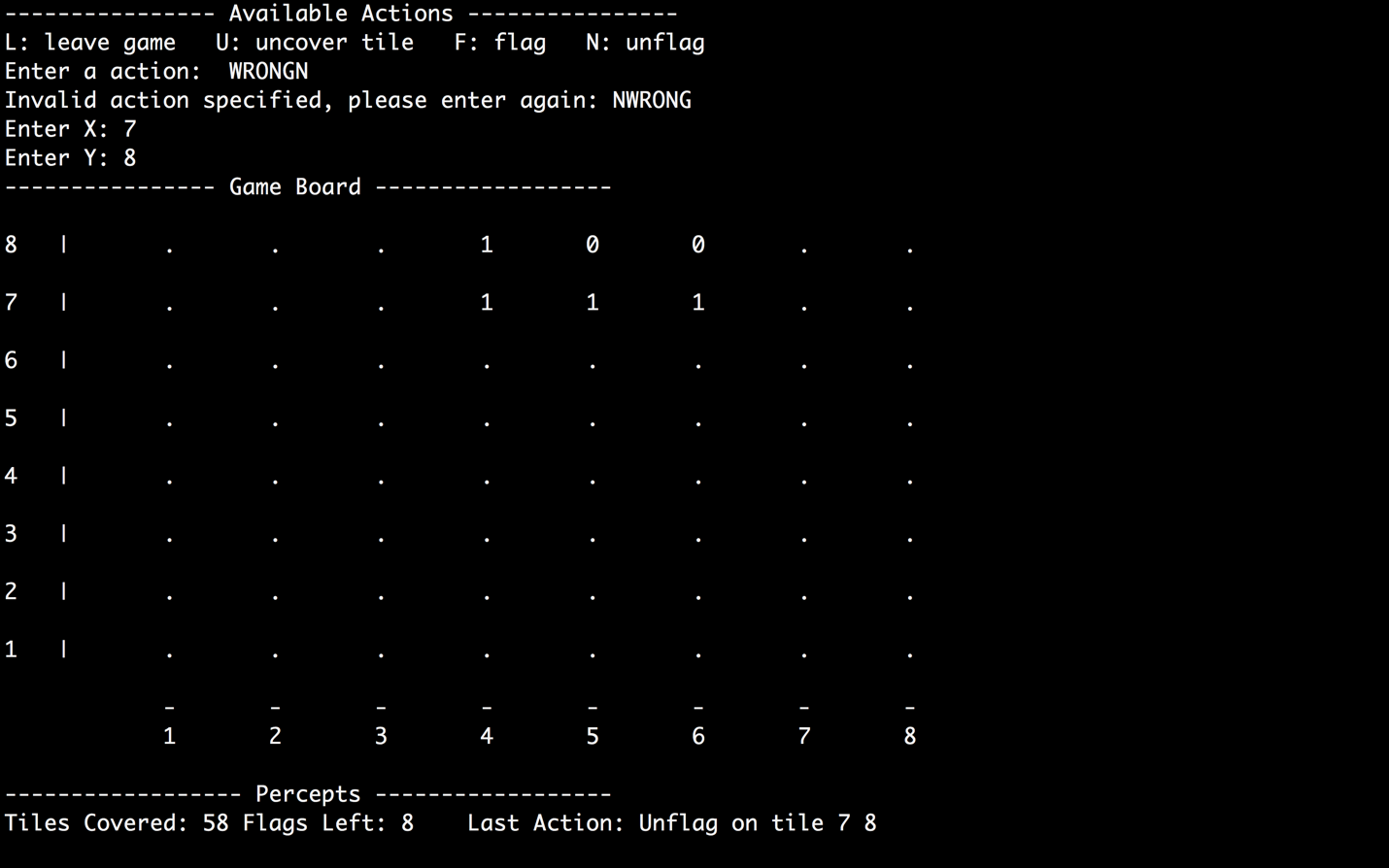
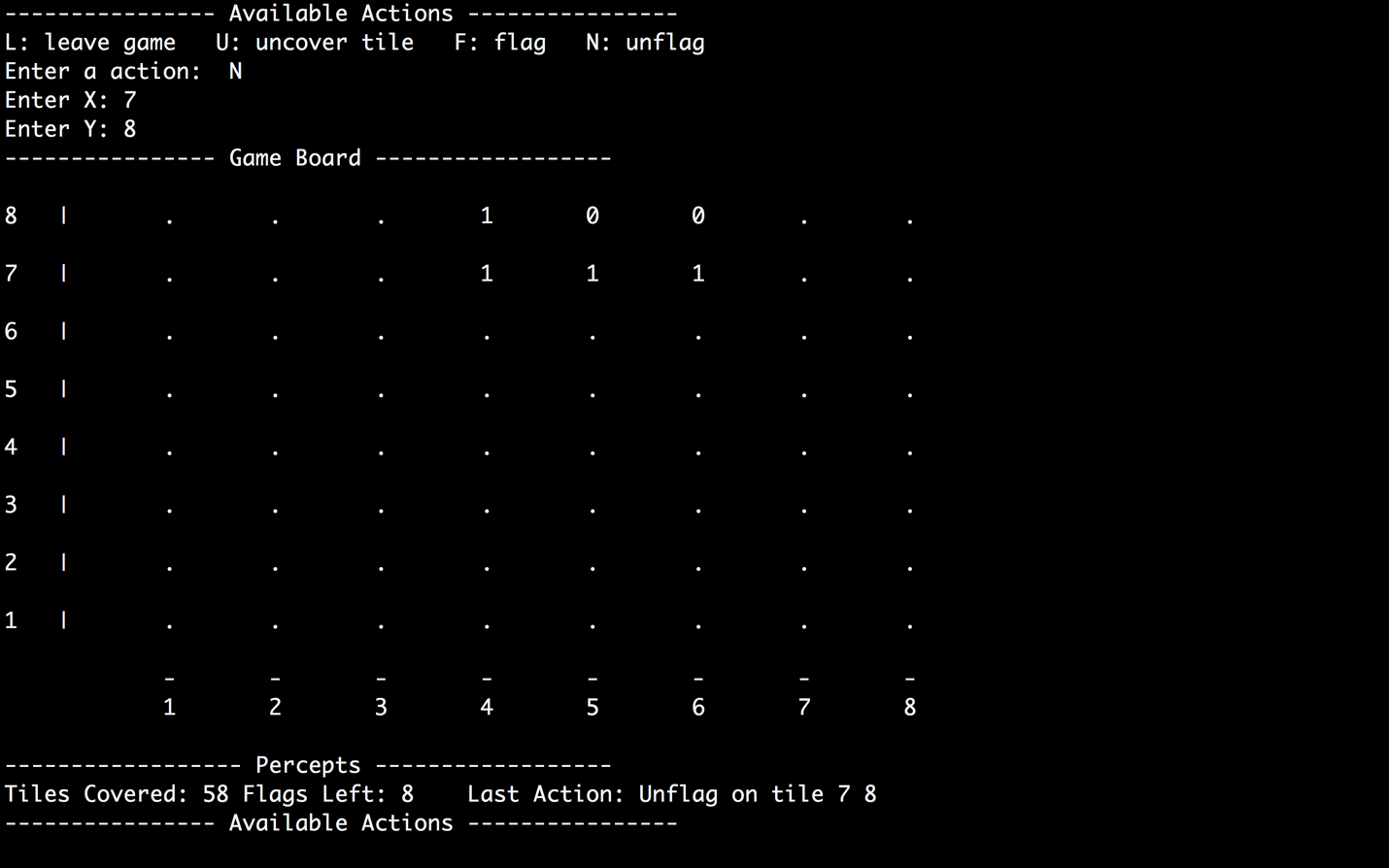
Flag on a tile that is already uncovered tile results no effect on the board displayed but this action actually reduces flags left from 10 to 9 and change flag Boolean of that tile to true. This scenario allows dumb agent to act irrationally.



The Manual AI accepts valid inputs as long as the first character or the first number is valid and it will discard whatever followed by the first valid input.



Uncover a flagged tile will actually uncover the tile and make the board to display the hint number, instead of flag symbol.



The raw input is Action{action\_type::LEAVE, -1, -1}

**Method Explanation**

This section defines all the methods in shell. Due to different languages used for shell, minor differences are expected.

**------------------------------------- File Format ---------------------------------------**

rowDimension colDimension

startSqX startSqY

2D board

*Example* *(8x8 board with 10 bombs and starting square (2,6)):*

8 8

2 6

0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 1 0

0 0 0 1 0 1 1 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 1 1 1

**------------------------------------------ Main -----------------------------------------**

* Read arguments
* Parse options
* Instantiate World
* Call world’s run method

**------------------------------------- World Class -------------------------------------**

**Attributes**

Int colDimension

Int rowDimension

Int score

Tile[][] board

Int totalMines

Int flagsLeft

Int coveredTiles

Private Class Tile

Bool mine

Bool covered

Bool flag

Int number

AI ai

**Public Methods**

World (String filename, String aiType, boolean debug)

1. If filename provided, construct board from file
   1. Instantiate/Initialize 2-D Tile Board with file content
2. Otherwise, create default board (9 x 9, w/ 10 mines, randomly pick start sq)
3. Call uncoverTile() on starting square
4. Initialize score to 0.
5. Instantiate the proper concrete AI object based on the aiType string passed and point “agent” instance variable to it.

Notes: In the above, use the addMines() and addNumbers() methods to populate board

run() : int score// engine function

1. while(not gameover)
   1. Get action from agent
   2. call doMove
2. Calculate Final Score
3. Call uncoverAll()

**Private Methods**

doMove(actionObj) : boolean

1. Get action from agent
2. Check if action is leave.
   1. If it is, return true (meaning “game over”)
3. Handle other actions accordingly

------------------------------------------ World Generating Functions --------------------------------------------

createBoard(File inputStream) : void

1. Creates 2-D tile array from first line of file and instantiates board instance variable

getFirstMove(File inputStream) : firstMove (depending on language)

1. Read the first move coordinates from file
   1. Do error check. Print error message (if error)
2. uncoverTile( first move coordinates).
3. Return coordinates

getFirstMove() : firstMove

1. Return random coordinate for first move
2. uncoverTile( first move coordinates).

addMines(File inputStream) : void

1. Reads the 0/1 grid from file and modify Tile attribute ‘mine’ for each

addMine(int row, int col): void

1. Add mine to board.
   1. Modify tile attribute

addNumbers() : void

1. Iterate entire board
   1. Increment tile “number” for each neighboring bomb

uncoverTile(int row, int col) : void

1) decrement “coveredTile” instance variable

2) set board[row][col].covered = false

uncoverAll() : void

1. Uncover all tiles

-------------------------------------------- World Printing Functions --------------------------------------------

**Format (This is a sample for ManualAI mode…fix the spacing when you implement)**

---------------- Game Board ------------------

6 | . . . . . . . .

5 | . . . . . . . .

4 | . . . . . . . .

3 | . . . . 0 . . .

2 | . F . . . . . .

1 | . . . . . . . .

- - - - - - - -

1 2 3 4 5 6 7 8

------------------ Percepts ------------------

Tiles Covered: 46 Flags Left: 16 Last uncovered Tile (number): 4,3

---------------- Available Actions ----------------

L: leave game U: uncover tile F: flag N: unflag

Enter a action: U

Enter x: 2

Enter y: 2

printWorld() : void

1. Call printBoardInfo
2. Call printAgentInfo

printBoardInfo() : void

1. Iterate board
   1. Call printTileInfo for each board coodinate

printAgentInfo() : void

1. Print percepts

// for debug mode and random ai mode

printActionInfo() : void

1. Check instance type of agent. If agent is ManualAI type, then print the “available actions” section (see above).

printTileInfo(int row, int col) : void

1. Check tile attributes and print accordingly. (i.e. is it flagged, covered...etc)

isInBounds( int row, int col) : void

1. Check coordinates are in bounds of board

**--------------------------- AI Class (abstract class) -------------------------------**

**Public Methods**

* public getAction(int number) : Action [abstract method]

Enum Action\_type { LEAVE, UNCOVER, FLAG, UNFLAG }

Inner class Action {

Action\_type action;

Int x;

Int y;

}

**-------------------------- MyAI Class (inherits from AI) --------------------------**

MyAI(rowDimension, colDimension, totalMines, startX, startY)

Implement getAction()

**----------------------- ManualAI Class (inherits from AI) -----------------------**

Implement getAction(int number)

1. Print “L: leave game U: uncover tile F: flag/unflag”
2. Print “Enter Action: “
3. Prompt for input
   1. Do error check on Action. Everything is invalid except for L, U, and F (must be capital)
4. Print “Enter X: “. Prompt user for input
   1. Check for valid X
5. Print “Enter Y: ”. Prompt user for input
   1. Check for valid Y
6. Put all steps above in a while loop.
7. Construct Action Object and return.
   1. Modify instance variables accordingly

**----------------------------- World Generator Script ------------------------------**

- createWorldFile(int nRows, int nCols, int difficulty, String filename) : String

Notes:

* Generated world must have a “patch” of non-mines around starting square.
* The number of bombs on the world depends on the difficulty:
  + Beginner (9x9, 10 mines),
  + Intermediate (16x16, 40 mines)
  + Expert (16x30, 99 mines)

**Scoring system and grading**

The scoring system that we devised is very simple. In general, the agent is awarded points only if the agent completes a world. Each difficulty awards a different amount of points: beginner is worth 1 point, intermediate is worth 2 points, and expert is worth 3 points.

While this system does not acknowledge partially solved worlds, it does measure an agent’s ability through the number of worlds completed. We felt that this was a more accurate metric for grading purposes. This type of scoring also encourages students to implement “smart guessing” in their agents. Instead of simply leaving the game, it is better for the agent to make a guess and risk dying since there is no penalty for death.

There are two forms of grading, both of which still use the same scoring system. One form grades on completion, which should be used to determine their project score. There are 3 project deadlines through the quarter that have increasing point requirements.

* Deadline 1 - This deadline is designed to allow the students familiarize with the project. This may include reading the source code, learning the functions, playing around with the code, and setting up the environment. The assignment should be assigned in week 3 and due by the end of week 5. **The formal requirement for this deadline is an agent that can solve 200/1000 “super easy” 5x5 with 1 mine worlds.**
* Deadline 2 - This deadline is designed to push students to develop a smarter agent with more capabilities, such as a form of guessing. This requirement now requires the agent to solve worlds that contain multiple mines. Previously, the super easy worlds only contained one mine, which usually results in little to no guessing at all. Now that the agent is working on multi-mined worlds, guessing is now required and smart forms of guessing are encouraged. **The formal requirement for this deadline is an agent that can solve 300/1000 beginner worlds and 150/1000 intermediate worlds.**
* Deadline 3 - This deadline is designed to encourage students to think of and implement more complex strategies. For example, in deadline 2, an agent might be able to get by with a guessing strategy that picks an arbitrary tile to uncover. This strategy can be improved by using a heuristic for guessing. **The formal requirement for this deadline is an agent that solve 500/100 beginner worlds and 350/1000 intermediate worlds.**

The project deadlines do not include any requirement on expert worlds because it is used in the other form of grading: points. The point grading system should be used in a tournament setting where students’ agents compete against each other for the highest point totals. The agents are then broken down into tiers, ordered by point totals.

The approximate max completions for each level is 700 out of 1000, 600 out of 1000, 100 out of 1000.

John’s Runs

Smart AI with random guessing

Easy: 558

Medium: 434

Hard: 27

Smart AI

Easy: 492

Medium: 384

Hard: 21

Dumb AI with random guessing

Easy: 456

Medium: 211

Hard: 1

Dumb AI

Easy: 383

Medium: 178

Hard: 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Easy | Medium | Expert | Score |
| Dumb AI | 383 | 178 | 1 | 742 |
| Dumb AI with guessing | 456 | 211 | 1 | 881 |
| Smart AI | 492 | 384 | 21 | 1323 |
| Smart AI with guessing | 558 | 434 | 27 | 1507 |

Justin’s Runs

(Dumb AI, does not guess and doesn’t do contradiction)

Easy: 397

Medium: 364

Hard: 16

Total: 1173

Jian runs

uncover safe only

easy: 0

med: 0

hard: 0

full version - guessing - contradiction’

easy: 383

med: 356

hard: 3

full version - guessing

easy: 489

med: 708

Hard: 42

full version

easy: 686

med: 578\*2 = 1156

hard: 74 \*3 =222

686 +1156+222 = 2,064

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scaled scores | Easy | Medium | Expert | Score |
| full version - guessing - contradiction’ | 383 | 178 \*2=356 | 1\*3 =3 | 642 |
| full version - guessing | 489 | 354\*2=708 | 14\*3=42 | 1,239 |
| full version | 686 | 578\*2 = 1156 | 74\*3= 222 | 2064 |