Henry Gorelick
CISC 6525
May 9, 2019

Wumpus World Project Report

Introduction

In the following sections, I will outline how my implementation of WWRationalAgent meets all four performance criteria, and therefore outperforms WWAgent in any dungeon.

Procedure

1. The *Simulation* object calls the agent's *self.update()* function and passes the percepts sensed at the agent's current position.
2. *WWRationalAgent.update()*
    1. If the agent senses "glitter," there's no need to update its knowledge because it will grab the gold in *WWRationalAgent.action()*.
    2. Sets *self.percepts* to a list of the not None values in the given percepts
    3. The agent marks its current cell as both safe and explored. This will contribute to fulfilling performance criteria #3.
    4. If there are no percepts at the agent's current position, then the agent iterates through the cells adjacent to its current position, and marks them all as safe. Thus fulfilling performance criteria #2: "*It always identifies whether a location is safe at the earliest time possible given percepts*." Furthermore, this will contribute to fulfilling performance criteria #1.
    5. Otherwise (there are percepts), the agent updates its Knowledge Base
        a. *WWRationalAgent.update_kb()*
            - If the agent sense "scream," then it has killed the Wumpus and can remove any "stench" percepts from its current position and from the Knowledge Base
            - The agent marks the current cell on the map as having each of its current percepts.
            - Then, the agent adds each percept to the Knowledge Base
3. The *Simulation* next calls the agent's *self.action()* function
4. *WWRationalAgent.action()*
    1. If the agent senses "glitter" it goes ahead and returns the "grab" action.
    2. If the agent's last move was a rotation, it returns an action based on *self.next_position*, which is saved whenever the agent chooses its move.
    3. If the agent has been to its last position more than 3 times, then it's safe to assume that it cannot find a safe path to the gold and the dungeon is unsolvable.
    4. Next, the agent calls *self.get_valid_moves()*
        a. For each cell adjacent to the agent's current position that hasn't been marked as safe:
            - If the agent perceives "stench", it calls *self.tt_entails(query)* with the query being "Wumpus at adjacent cell"

- If the agent perceives "stench", it calls *self.tt_entails(query)* with the query being "Pit at adjacent cell"

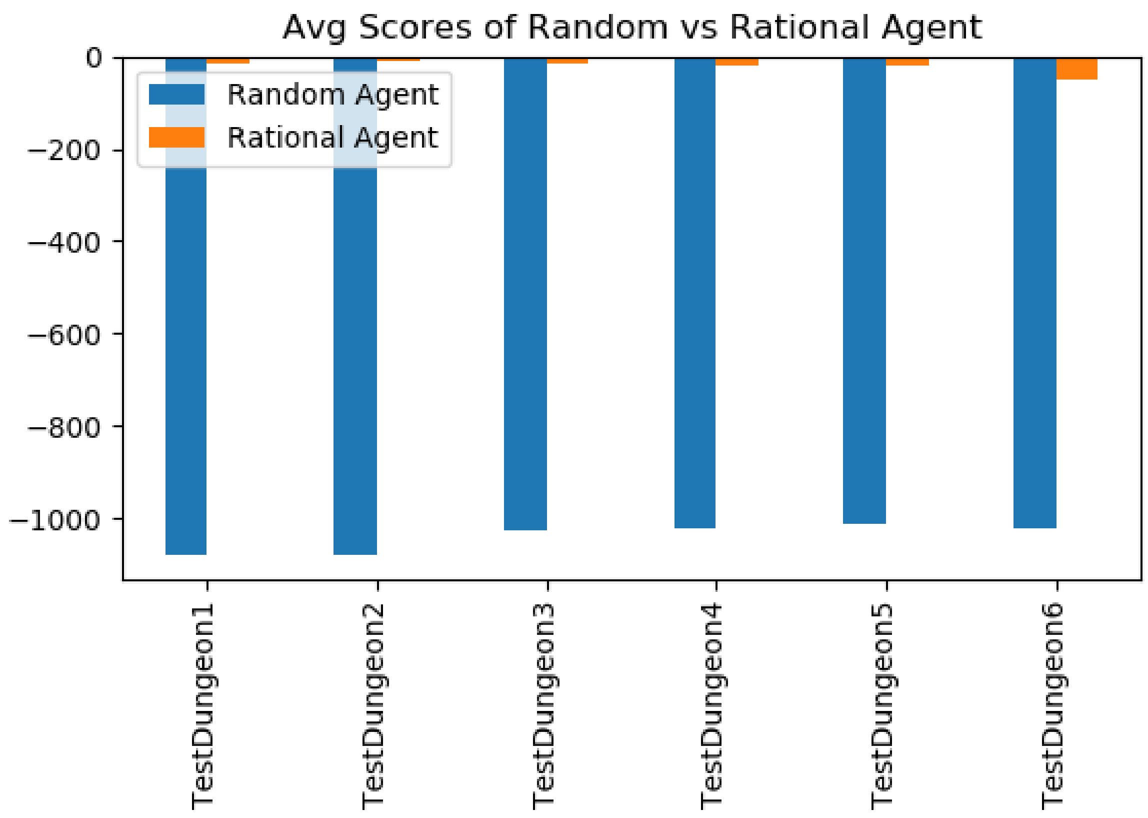
b. If either enumeration of the truth table with the given query returns true, that adjacent cell is added to a list of bad moves.

c. Finally, the function returns a list of adjacent cells with the bad cells filtered out. Thus fulfilling performance criteria #1.

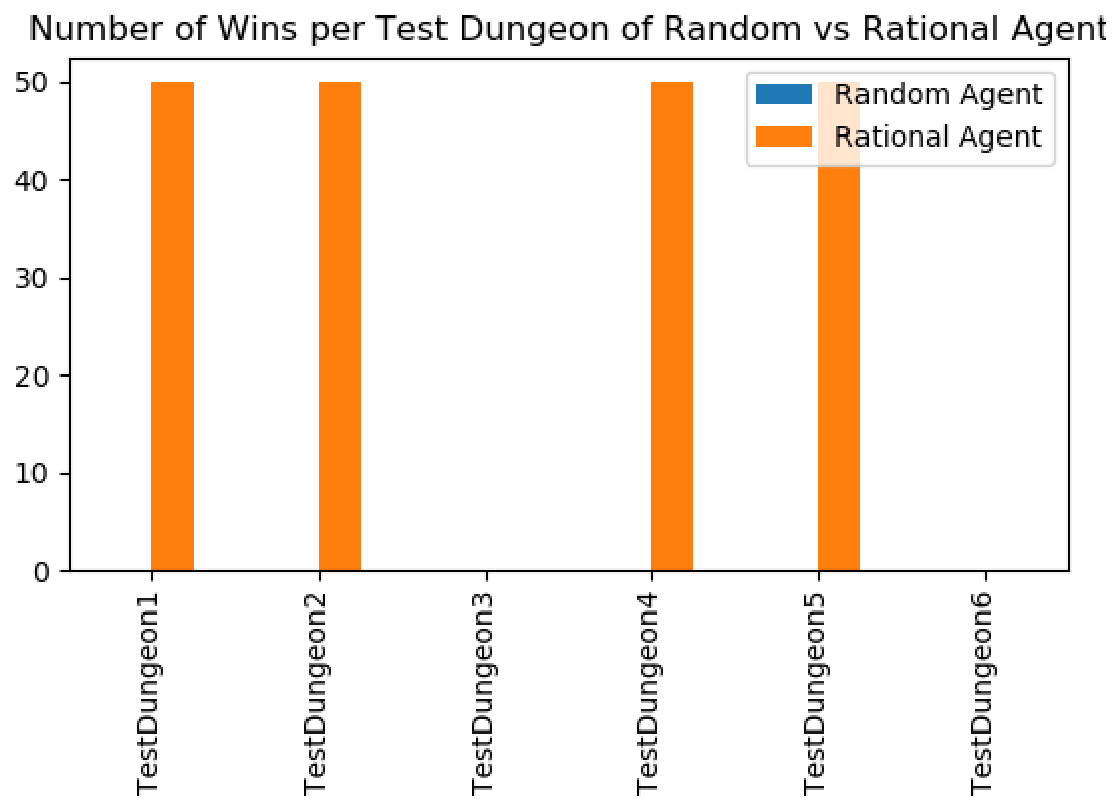
5. If there are no valid moves, the dungeon is not solvable, and the agent exits.
6. With the list of valid moves, the agent creates a list of the unexplored cells in the list of valid moves.
7. If there are unexplored cells, the agent chooses to move to the first of the unexplored cells.
8. Otherwise, the agent chooses to move to a cell its been to the fewest amount of times.
9. Finally, the agent gets the appropriate action based on its current position, the direction its currently facing, and the returned next position, and returns the action to the *Simulation*.

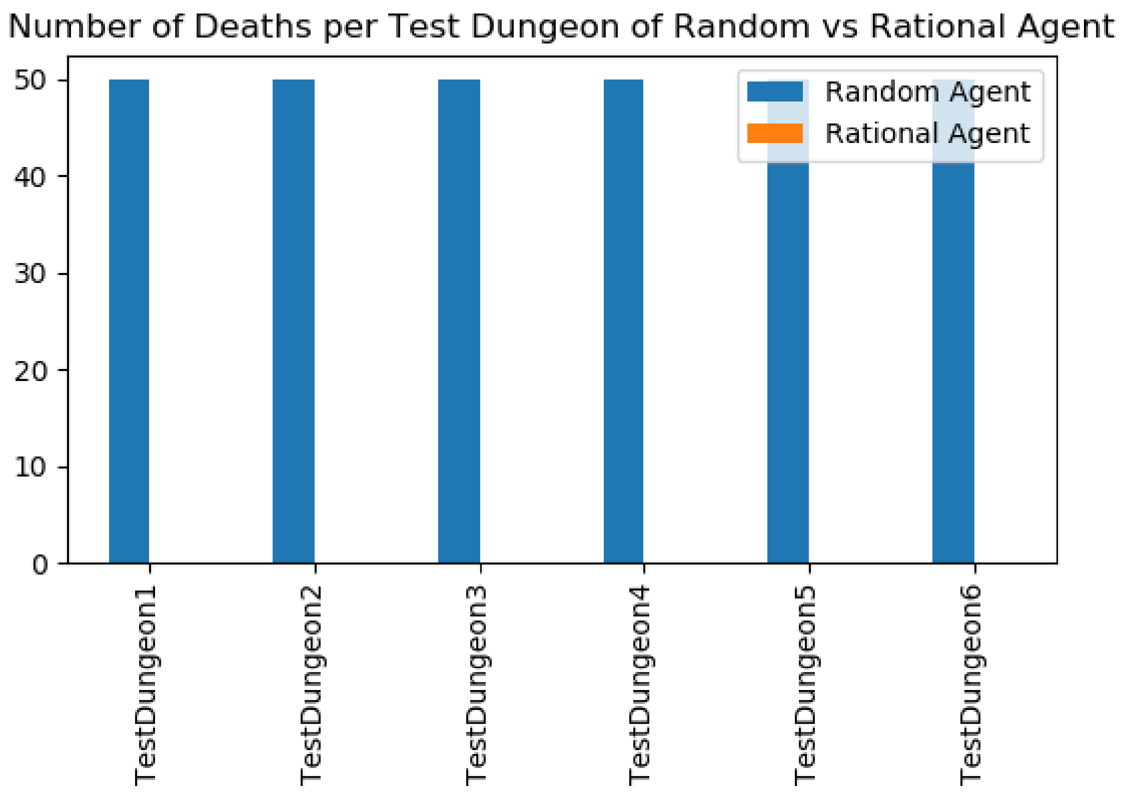
Performance Comparison

The *Simulation* object has a built-in *score* value and score calculation function. Therefore, it is simple to compare the performance of WWRationalAgent to WWAgent. The charts below depict score comparisons for 50 iterations of 6 test dungeons, as well as a few other metrics.



Henry Gorelick
CISC 6525
May 9, 2019


Number of Wins per Test Dungeon of Random vs Rational Agent
Random Agent
Rational Agent
TestDungeon1
TestDungeon2
TestDungeon3
TestDungeon4
TestDungeon5
TestDungeon6


Number of Deaths per Test Dungeon of Random vs Rational Agent
Random Agent
Rational Agent
TestDungeon1
TestDungeon2
TestDungeon3
TestDungeon4
TestDungeon5
TestDungeon6