CISC 6525 Artificial Intelligence

Final Project - Wumpus World

Li Yihao
12/13/2020

Abstract: This code *wwagent.py* is an implementation of the agent in the Wumpus World question. The agent was designed to intelligently choose the next action in the Wumpus World in order to win the game (have the gold) or leave the cave safely. This code should work with the *wwsim.py* file provided by Greg Scott's GitHub. This code was modified from the *wwagent.py* file provided by the CISC6525 class which only choose random actions and was also edited from Greg Scott's file.

Part I. Feature Design

In this code, the agent was designed to intelligently pick the next action in the Wumpus World in order to win or leave safely. Specifically, it was expected to do the following:

1) Agent will pick a 100% safe move from its current position if it exists.
2) If there's no such move, agent will try to kill the Wumpus if it knows the Wumpus' location and also the location is 100% free from a pit.
3) Agent will pick the safest move if former two are not available.
4) If gold is reachable, and agent isn't killed because of step 3 above, then agent will eventually grab it, climb out, and win the game.
5) If gold isn't reachable, and agent isn't killed because of step 3 above, then agent will climb out safely without the gold.
6) Agent always identifies the safety of edge cells (accessible cells that the agent hasn't been to) as early as possible given percepts.

Part II. Implementation Explained

1. Major algorithms:

In this code, I mainly implemented three different AI methods to achieve the designed features of the agent. The methods and related functions are:

1) **propositional logic**: `self.isTrue(prop,model)`
2) **truth table enumeration**: `self.model_pos(symbols,model,KB,qry)`, `self.model_pos_wp(symbols,model,KB,qry)`
3) **uniform cost search (a variation)**: `self.search_route(start,goal)`

I will explain how I used these algorithms in detail later in the function explaining part.

2. Workflow:

In order to achieve the expectation, a robust workflow is also needed to implement the algorithm at every step.

The main idea is that every time the agent gets into a new cell or status, it should calculate a plan and remember that plan until the it is successfully carried out. And while there is a plan, the agent doesn’t need to calculate a new one, it just needs to calculate the next action to finish the plan.

The detailed workflow is as follow:

Step 0. Deal with the Gold

Step 0.1 Grab the gold if the agent sees the glitter.

Step 0.2 If the agent has the gold now, generate the route to climb out of the cave.

Step 1. Update the moving plan

Delete the first cell in self.__route (the moving plan) if current position is the first cell.

Step 2. Update knowledge base, model, and make a new plan

If the agent doesn't have a plan currently, which means that the agent is in a new cell or it just killed the Wumpus, the agent needs to update KBs and models. Then agent will make a new plan based on the new information.

Step 2.1 Update the knowledge bases and models.

Step 2.2 Add adjacent cells to self.__edge (a list of edge cells, which are accessible but the agent hasn't been to) if they are not in self.__beento (a list of cells that the agent has been to) and not in self.__edge.

Step 2.3 Update the probability of being safe for every cell in self.__edge. There are three probabilities for each cell - P of free from the Wumpus (p_nw), P of free from a pit (p_np), and P of free from both (p_safe).

Step 2.4 Now the agent can make a new plan!

Step 2.4.1 If the largest p_safe < 1, check whether the Wumpus is good to be killed. When we know a certain cell has the Wumpus in it and the cell does not have a pit in it. Activate the hunting plan, and also choose the Wumpus cell as the destination cell.

Step 2.4.2 If not having a hunting plan, choose a safe cell (p_safe = 1) or the possibly safest cell (0 < p_safe < 1) as the destination cell, or if there are only unsafe cells (p_safe = 0), activate the plan to climb out and set the sell (0, 3) to be the destination cell.

Step 2.4.3 Whatever the new plan is, search for a route to the destination cell. If we are having a hunting plan now, remember to delete the last cell in our plan route because we don't want to step in that cell!

Step 3. Carry out the current plan

Since we are having a plan currently, calculate next action base on our plan.

Step 3.1 If we have a route now, go to the next cell in our route.

Step 3.2 Otherwise, if we have the hunting plan, shoot an arrow to the Wumpus.

Step 3.3 Otherwise, we must want to exit!

Step 4. Wind-up

Calculate new position/direction/arrow status basing on the action. Print out action and new status of the agent (if you want).

3. New/Revised Variables:

`self.__WumpusAlive` - Wumpus alive or not

`self.__arrow` - arrow available or not

`self.__kb_w` - knowledge base for Wumpus related information

`self.__kb_p` - knowledge base for store pit information `self.__model_w` - model for Wumpus related information

`self.__model_p` - model for pit related information

`self.__beento` - all cells that agent has already been to

`self.__edge` - all accessible cells that agent hasn't been to

`self.__route` - the planned route, also the moving plan

`self.__hunt` - the hunting plan (put in the Wumpus cell if activated)

`self.__climbout` - the exiting plan (True if activated)

`self.__HaveGold` - REVISED from `self.__stopTheAgent`, True if having gold

4. New/Revised Functions:

`self.find_adj_cells(c_c)`

Generate a tuple of all adjacent cells given the current cell.

`c_c` (current cell) must be given as a string of x and y coordinate, eg. "xy"

If `c_c` is not given, will use `self.__position` to generate current cell.

`self.isTrue(prop, model)`

Recursive function that checks whether a set of propositional logic sentences (`prop`) is True or False given the provided semantics (`model`).

`model` is a dictionary using symbols as keys and storing the True/False in value.

e.g. `model` = {"p1":True, "p2":False, "p3":False}

`prop` represents logic sentences in the form of a nested list, prop can be either a query or a knowledge base.

e.g. `prop` = [[[["not", "p1"], "and", "p2"]], "and", ["p2", "implies", "p3"]]

`self.gen_nkb(okb, ps, new_s)`

Add a new sentence to current knowledge base and return the new knowledge base.

The form of a knowledge base is the same as that of a prop in `self.isTrue(prop, model)`.

`self.update_kb_model()`

Knowledge should be recorded whenever the agent goes into a new cell. This function updates the status of the Wumpus (alive or not) and record the sentences and semantics about Wumpus, pit, stench, and breeze of the current cell and adjacent cells.

The form of a knowledge base is the same as that of a prop in **`self.isTrue(prop, model)`**.

The form of a model is the same as that of a model in **`self.isTrue(prop, model)`**.

`self.model_pos(symbols, model, KB, qry)`

Check if a knowledge base (**`KB`**) entails a query (**`qry`**) by model enumeration. It returns the P (probability) of the **`qry`** being True under the circumstances that the **`KB`** is True, given all the possible models.

If returns $P = 1$, **`KB`** entails **`qry`**.

If returns $P < 1$, **`KB`** does not entail **`qry`**.

If returns $P = 0$, **`KB`** entails (**`not qry`**).

The form of **`KB`** and **`qry`** are the same as that of a **`prop`** in **`self.isTrue(prop, model)`**.

model stores all the known semantics.

`symbols` is a list of symbols with unknown semantics to be enumerated.

`self.model_pos_wp(symbols, model, KB, qry)`

A variation of **`self.model_pos(symbols,model,KB,qry)`** customized for Wumpus. Since the Wumpus exists in one and only one cell, the enumeration method is changed in this function.

`self.flatten(nest_iter)`

Flatten a nested list/tuple by yielding all items if it is not a list or tuple. This function makes it easier for us to find needed symbols from a KB.

`self.check_pit(cell)`

Use pit related knowledge base (**`self.__kb_p`**) and model (**`self.__model_p`**) to generate the P (probability) of a given **`cell`** being free from a pit. Besides, update **`self.__kb_p`** and **`self.__model_p`** when a pit is certain to exist (P=0) or not exist (P=1) in that **`cell`**.

`self.check_wumpus(cell)`

Use Wumpus related knowledge base (`self.__kb_w`) and model (`self.__model_w`) to generate the P (probability) of a given `cell` being free from Wumpus. Besides, update `self.__kb_w` and `self.__model_w` when Wumpus is certain to exist (P=0) or not exist (P=1) in that `cell`.

`self.search_route(start, goal)`

Search a best route (least actions) from the starting cell (`start`) to the goal cell (`goal`). It's a variation of UCS (uniform cost search). The step cost is 2 when requiring a turn, otherwise 1.

The route is given with a list of cells (from the next cell to the goal cell). All the cells must be selected from `self.__beento` (meaning it's safe) except for the goal cell.

`self.cal_action(current_goal, act_type)`

Given current cell, current direction, and adjacent cell, calculate and return the next action to move/shoot into that adjacent cell.

`act_type` should be "move" or "shoot", defaulted to be "move".

`self.action()`

This function is the major function to return the action. It follows the workflow mentioned earlier and utilizes all the functions above to calculate the next action.

Part III. Results and Evidence

Comparing to the original random agent (I will use **RA** to refer to this agent), this new agent (I will use **IA** to refer to this agent) is able to convey a much better performance on average multiple repetitions.

Baseline Performance

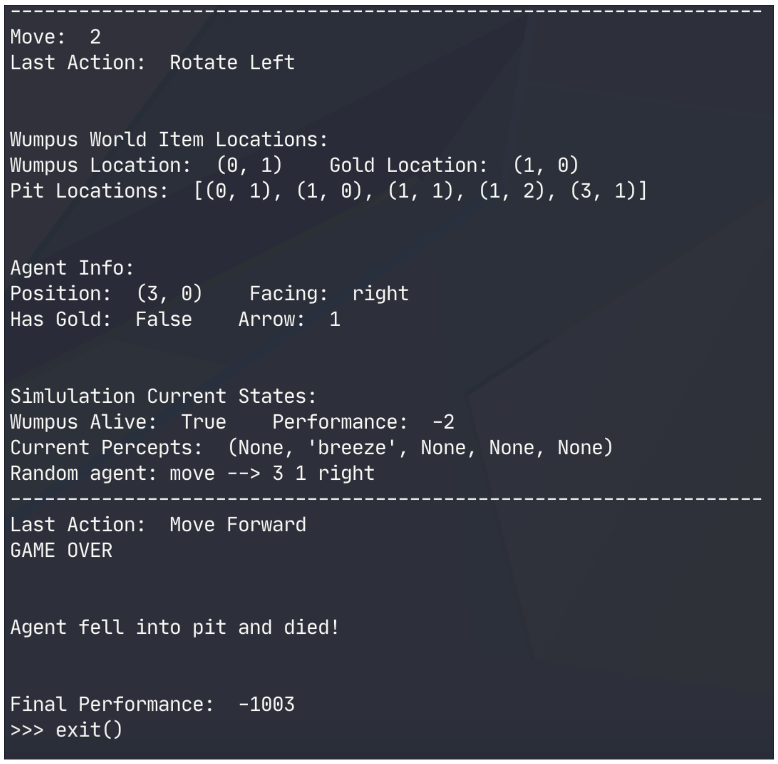
In order to build a baseline, I first ran the Wumpus World simulation 30 times using **RA**.
As we can see from Table 1, the **RA** ended up with dying 27 times out of 30 times. It only got the gold third. Using these results, we can easily calculate that the average count of steps before

episode ending is 12.8, the average count of steps before agent dying is also 12.8, and the average count of steps before agent getting the gold is 13.
There are no significant differences between different kind of results. Verifying that whatever the outcome is, this agent is just picking a step randomly. The three times of getting the gold is just pure luck!

Table 1. The results of 30 times of simulations using the random agent

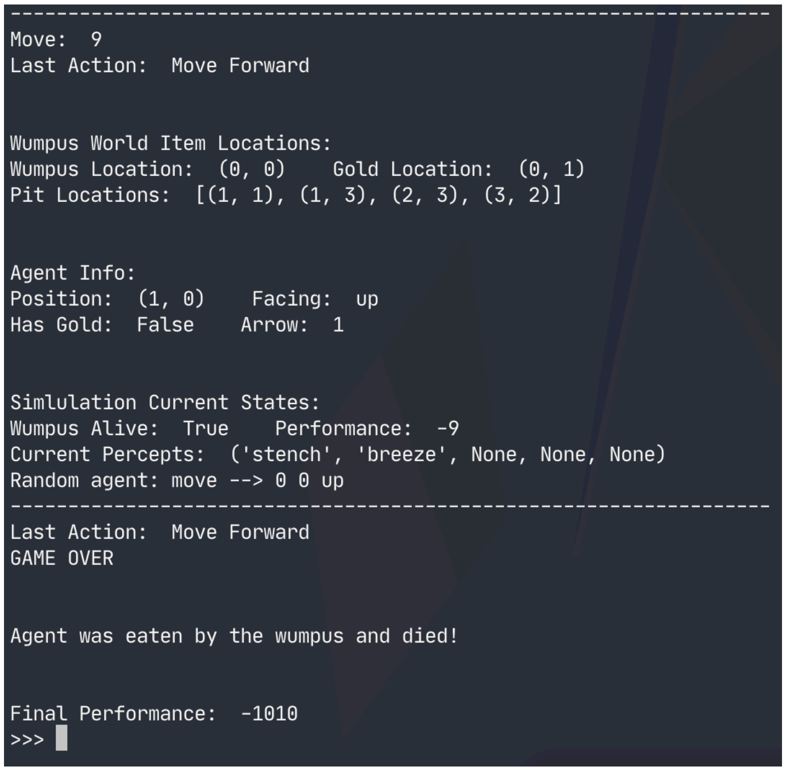
Result	Count	Ave Step
Fall into Pit	20	12.4
Eaten by Wumpus	7	13.9
Have the Gold	3	13

Figure 1 to Figure 3 shows the different results of simulations with **RA**.



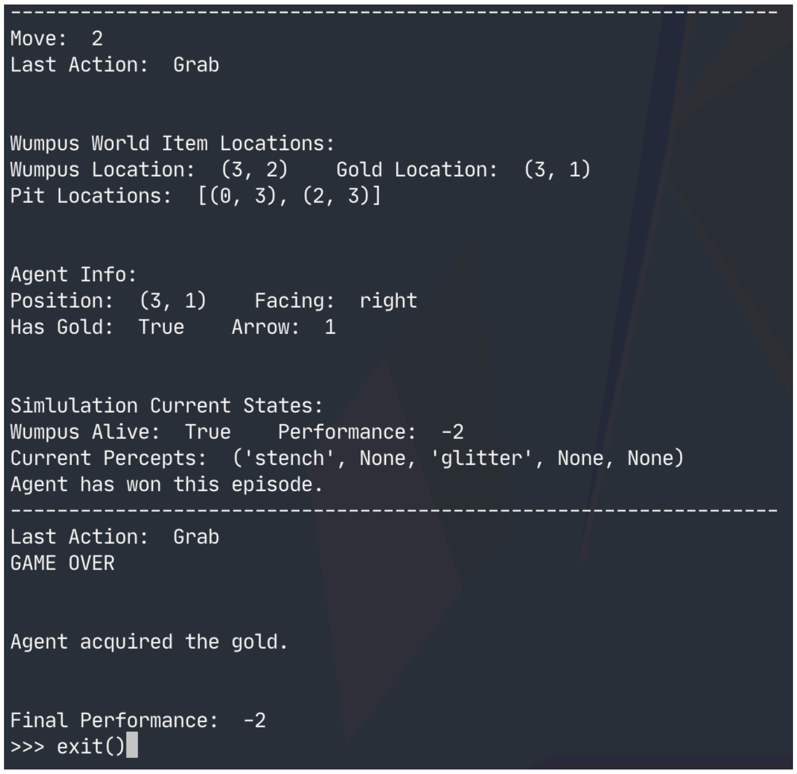
```
-----------------------------------------------------------------
Move:  2
Last Action:  Rotate Left


Wumpus World Item Locations:
Wumpus Location:  (0, 1)    Gold Location:  (1, 0)
Pit Locations:  [(0, 1), (1, 0), (1, 1), (1, 2), (3, 1)]


Agent Info:
Position:  (3, 0)    Facing:  right
Has Gold:  False    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  -2
Current Percepts:  (None, 'breeze', None, None, None)
Random agent: move --> 3 1 right
-----------------------------------------------------------------
Last Action:  Move Forward
GAME OVER


Agent fell into pit and died!


Final Performance:  -1003
>>> exit()
```

Figure 1. Random Agent fell into a pit and died



```
-----------------------------------------------------------------
Move:  9
Last Action:  Move Forward


Wumpus World Item Locations:
Wumpus Location:  (0, 0)    Gold Location:  (0, 1)
Pit Locations:  [(1, 1), (1, 3), (2, 3), (3, 2)]


Agent Info:
Position:  (1, 0)    Facing:  up
Has Gold:  False    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  -9
Current Percepts:  ('stench', 'breeze', None, None, None)
Random agent: move --> 0 0 up
-----------------------------------------------------------------
Last Action:  Move Forward
GAME OVER


Agent was eaten by the wumpus and died!


Final Performance:  -1010
>>> 
```

Figure 2. Random Agent was eaten by the Wumpus and died



```
-----------------------------------------------------------------
Move:  2
Last Action:  Grab


Wumpus World Item Locations:
Wumpus Location:  (3, 2)    Gold Location:  (3, 1)
Pit Locations:  [(0, 3), (2, 3)]


Agent Info:
Position:  (3, 1)    Facing:  right
Has Gold:  True    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  -2
Current Percepts:  ('stench', None, 'glitter', None, None)
Agent has won this episode.
-----------------------------------------------------------------
Last Action:  Grab
GAME OVER


Agent acquired the gold.


Final Performance:  -2
>>> exit()
```

Figure 3. Random Agent acquired the gold

Performance of the Intelligent Agent

IA ended up with dying for only 12 times, with an average step count of 8.6. It got the gold and left the cave safely for 17 times, averaging a step count of 19.8. Besides, **IA** also left the cave safely without the gold once with a step count of 43. It was because in that simulation the gold is unreachable, and the **IA** decided to leave since all the remaining cells were unsafe.

Table 2. The results of 30 times of simulations using the intelligent agent

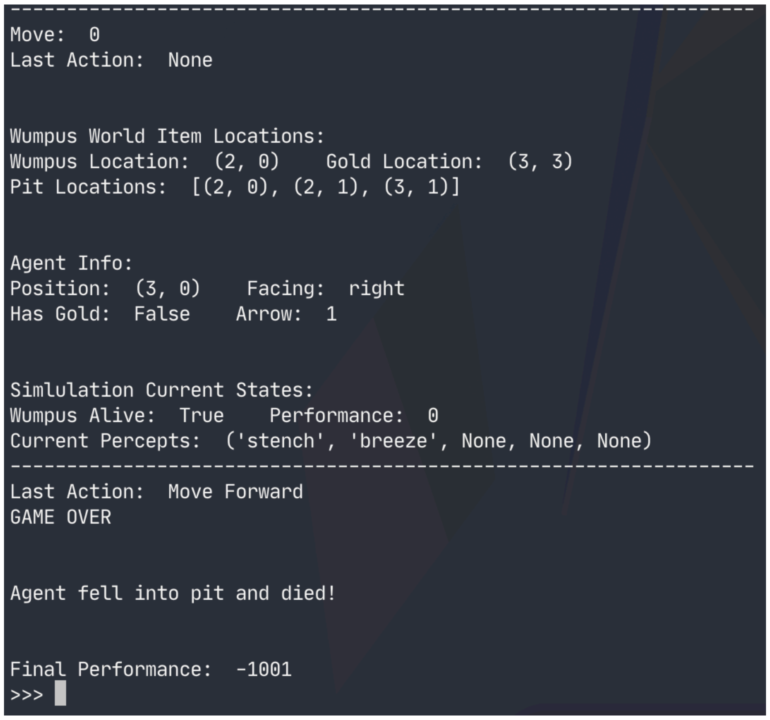
Result	Count	Ave Step
Fall into Pit	8	11.1
Eaten by Wumpus	4	3.5
Exit with Gold	17	19.8
Exit without Gold	1	43

We notice that the average step counts for “Fall into Pit” and “Eaten by Wumpus” are lower than that for the **RA** simulations. This is because the **IA** is more likely to be killed at the earlier stage of a game due to the lack of information, especially for the case of “Eaten by Wumpus”. Since there is one and only one Wumpus in a game, and the agent is able to kill the Wumpus, it is easier for the agent to locate or kill the Wumpus than to locate every pit.

To get the gold, **IA** sometimes had to explore the cave for a long time. Other times, it was able to find the gold quickly out of luck. And to exit without the gold, **IA** have to almost entirely travel the cave. In both cases, **IA** also need to go back to the starting cell and climb out. That’s why exiting with or without the gold takes **IA** much more steps.

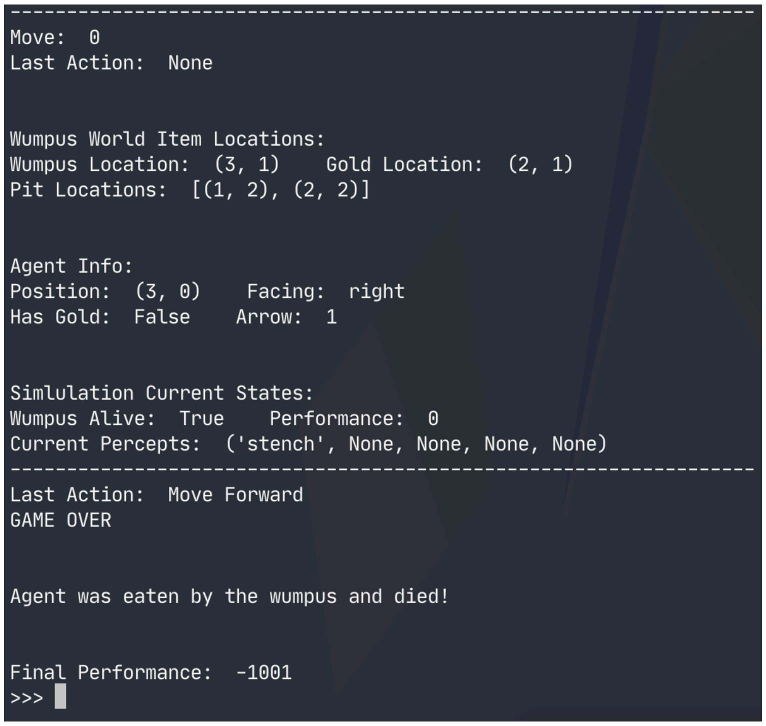
Figure 4 to Figure 7 shows the different results of simulations with **IA**.

To summary, the intelligent agent reduces the dying rate from **90%** to **40%** and increases the winning rate from **10%** to **56.7%**. It is even able to avoid dying and leave the cave safely when the gold is not accessible in some cases. Comparing with the baseline performance by the random agent, the intelligent agent given by this code is far better.



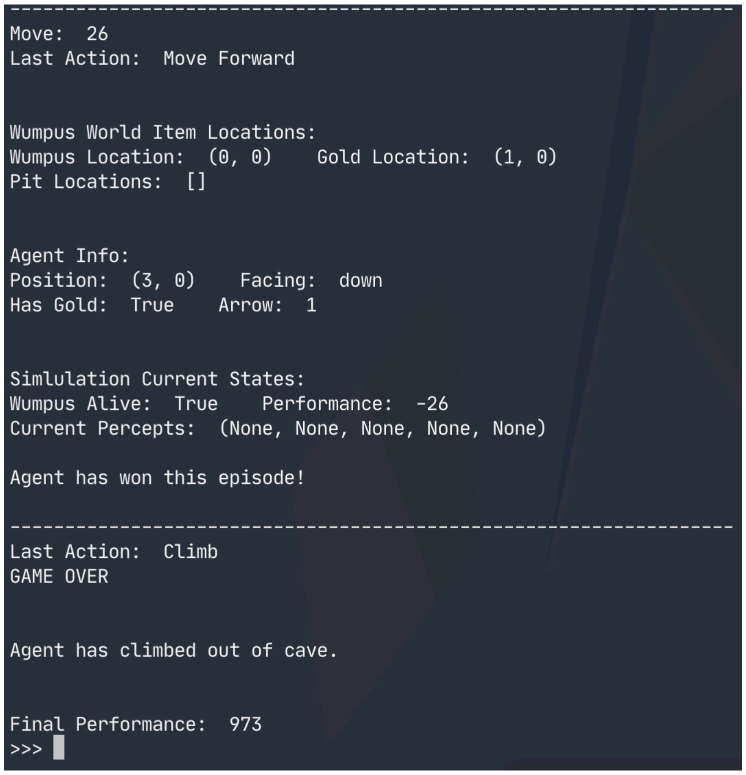
```
-------------------------------------------------------------------
Move:  0
Last Action:  None


Wumpus World Item Locations:
Wumpus Location:  (2, 0)    Gold Location:  (3, 3)
Pit Locations:  [(2, 0), (2, 1), (3, 1)]


Agent Info:
Position:  (3, 0)    Facing:  right
Has Gold:  False    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  0
Current Percepts:  ('stench', 'breeze', None, None, None)
-------------------------------------------------------------------
Last Action:  Move Forward
GAME OVER


Agent fell into pit and died!


Final Performance:  -1001
>>> 
```

Figure 4. Intelligent Agent fell into a pit and died



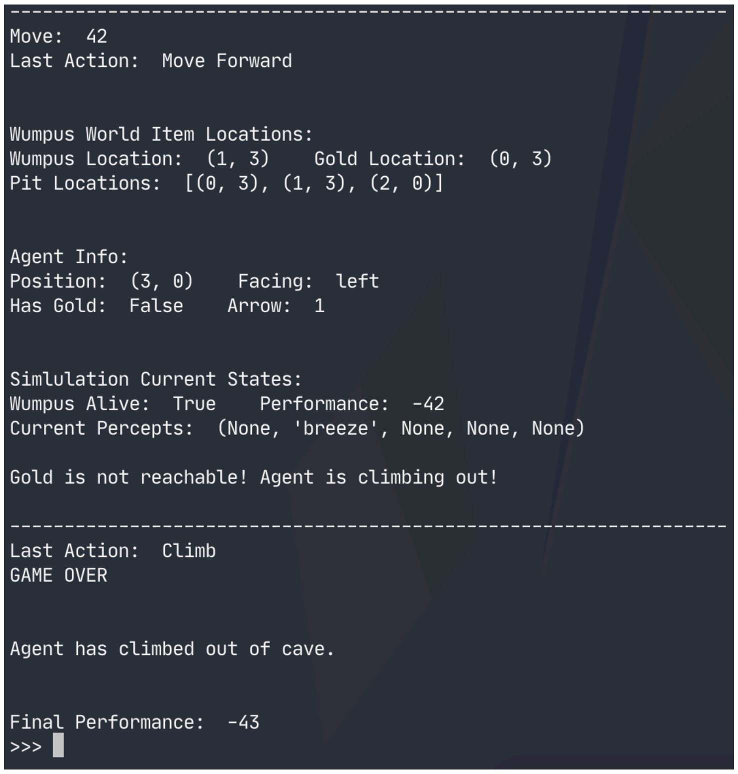
```
-------------------------------------------------------------------
Move:  0
Last Action:  None


Wumpus World Item Locations:
Wumpus Location:  (3, 1)    Gold Location:  (2, 1)
Pit Locations:  [(1, 2), (2, 2)]


Agent Info:
Position:  (3, 0)    Facing:  right
Has Gold:  False    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  0
Current Percepts:  ('stench', None, None, None, None)
-------------------------------------------------------------------
Last Action:  Move Forward
GAME OVER


Agent was eaten by the wumpus and died!


Final Performance:  -1001
>>> 
```

Figure 5. Intelligent Agent was eaten by the Wumpus and died



```
-------------------------------------------------------------------
Move:  26
Last Action:  Move Forward


Wumpus World Item Locations:
Wumpus Location:  (0, 0)    Gold Location:  (1, 0)
Pit Locations:  []


Agent Info:
Position:  (3, 0)    Facing:  down
Has Gold:  True    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  -26
Current Percepts:  (None, None, None, None, None)

Agent has won this episode!

-------------------------------------------------------------------
Last Action:  Climb
GAME OVER


Agent has climbed out of cave.


Final Performance:  973
>>> 
```

Figure 6. Intelligent Agent exited the cave with the gold



```
-------------------------------------------------------------------
Move:  42
Last Action:  Move Forward


Wumpus World Item Locations:
Wumpus Location:  (1, 3)    Gold Location:  (0, 3)
Pit Locations:  [(0, 3), (1, 3), (2, 0)]


Agent Info:
Position:  (3, 0)    Facing:  left
Has Gold:  False    Arrow:  1


Simlulation Current States:
Wumpus Alive:  True    Performance:  -42
Current Percepts:  (None, 'breeze', None, None, None)

Gold is not reachable! Agent is climbing out!

-------------------------------------------------------------------
Last Action:  Climb
GAME OVER


Agent has climbed out of cave.


Final Performance:  -43
>>> 
```

Figure 7. Intelligent Agent climbed out of the cave without the gold