



**Ayoub Mabkhout**

**Teammate: Chifaa Bouzid**

**Instructor: Dr. Tajjeeddine Rachidi**

**March 21, 2022**

**CSC 4301: Intro. to Artificial Intelligence Project #2**

**Wumpus World Knowledge-Based Intelligent Agent with  
Declarative Knowledgebase Implementation**

## **Table of Contents**

**Introduction**

**Program Structure**

**Predicates**

**Successful Case**

**Failed Case**

**Agent Weaknesses**

**Conclusion**

## Introduction

This is a report detailing a program for the Wumpus World game with an Intelligent Agent that solves the game.

The rules of the game are as detailed in the following link:

<https://www.javatpoint.com/the-wumpus-world-in-artificial-intelligence>

For the purposes of this assignment, the goal for the intelligent agent is to kill the Wumpus. It may pick up the gold if it happens to find it on its way to kill the Wumpus.

The intelligent agent *\*dies\**, meaning that it reaches a state of game over, if it enters a room with a pit in it or if it enters the Wumpus room while it is still alive.

## Program Structure

The program was coded using two languages, the first being Prolog and the second being Java. First, the Prolog part represents the knowledgebase consulted by the intelligent agent. The knowledgebase contains all of the predicates necessary to make logical inferences about the presence of a pit, a Wumpus, or the presence of gold based on the sensory information given by the intelligent agent's actuators. It also has a rule called 'bestAction(X)' that can be queried by the agent and that returns the best action that will be performed by the intelligent agent. The predicates will be explored in more detail in the next section.

The Java part of the program contains the information relating to the world map. It has a class with a method that randomly generates a world where pits are distributed with a probability of 20%, and where there is only one Wumpus and one Gold Treasure.

The world is displayed on a Java GUI.

The program also contains the game loop which updates the game state after each action performed by the intelligent agent, namely, the GUI is updated to reflect the corresponding changes.

Finally, it contains an intelligent agent that consults the knowledge base using the Tell() and Ask() APIs programmed using the JPL package.

### Key Predicates

```
adjacent(X1,Y1,X2,Y2) :-
    room(X1,Y1),
    room(X2,Y2),
    X1 is X2,
    Y1 is Y2 + 1.

adjacent(X1,Y1,X2,Y2) :-
    room(X1,Y1),
    room(X2,Y2),
    X1 is X2,
    Y1 is Y2 - 1.

adjacent(X1,Y1,X2,Y2) :-
    room(X1,Y1),
    room(X2,Y2),
    X1 is X2 + 1,
    Y1 is Y2.

adjacent(X1,Y1,X2,Y2) :-
    room(X1,Y1),
    room(X2,Y2),
    X1 is X2 - 1,
    Y1 is Y2.

adjacentrooms(room(X,Y),Rooms) :-
    findall(room(X2,Y2),adjacent(X,Y,X2,Y2),Rooms).
```

The predicate adjacent/4 is used to check if two rooms are adjacent or to return all rooms adjacent to each other. X1 and Y1 are the coordinates of one room, and X2 and Y2 are the coordinates of a second room.

```
potentialPit(X,Y) :-
    room(X,Y),
    not(notAPotentialPit(room(X,Y))).
```

```
potentialWumpus(X,Y) :-
    room(X,Y),
    not(notAPotentialWumpus(room(X,Y))).
```

```
safe(room(X,Y)) :-
    not(potentialPit(X,Y)),
    not(potentialWumpus(X,Y)).
```

potentialPit/2 and potentialWumpus/2 both are predicates that tell if a room may have a pit or a Wumpus is the tested room. Returning false means that there is no pit or Wumpus for certain, implying that the room is safe to move to. Returning true only means that the result is inconclusive; the room may be safe, but given the limited and very local information, the agent cannot always determine that.

```
notACertainPit(X,Y,X2,Y2):-
    room(X3,Y3),
    adjacent(X2,Y2,X3,Y3),
    X3 \= X,
    Y3 \= Y,
    potentialPit(X3,Y3).

certainPit(X,Y,X2,Y2):-
    not(notACertainPit(X,Y,X2,Y2)).

pit(X,Y):-
    room(X,Y),
    potentialPit(X,Y),
    room(X2,Y2),
    adjacent(X,Y,X2,Y2),
    breeze(room(X2,Y2)),
    certainPit(X,Y,X2,Y2).
```

```
notACertainWumpus(X,Y,X2,Y2):-
    room(X3,Y3),
    adjacent(X2,Y2,X3,Y3),
    X3 \= X,
    Y3 \= Y,
    potentialWumpus(X3,Y3).

certainWumpus(X,Y,X2,Y2):-
    not(notACertainWumpus(X,Y,X2,Y2)).

wumpus(X,Y):-
    room(X,Y),
    potentialWumpus(X,Y),
    room(X2,Y2),
    adjacent(X,Y,X2,Y2),
    stench(room(X2,Y2)),
    certainWumpus(X,Y,X2,Y2).
```

pit/2 and wumpus/2 are the predicates that tell for certain if there is a pit or a Wumpus. Either of them returning true means that the agent knows for a fact that there is a pit or a Wumpus in the given location (X,Y).

```
gold(room(X,Y)):-
    glitter(room(X,Y)).
```

```
grabGold(room(X,Y)):-
    hunter(room(X,Y)),
    gold(room(X,Y)).
```

There is gold in a room if the agent senses glitter in that room. The agent can pick up gold if it happens to be in a room with gold in it.

```

connected(room(X,Y),room(Xt,Yt)) :-
    adjacent(X,Y,Xt,Yt) .

path(room(X,Y),room(Xt,Yt),Path) :-
    travel(room(X,Y),room(Xt,Yt),[room(X,Y)],Q),
    reverse(Q,Path) .

travel(room(X,Y),room(Xt,Yt),P,[room(Xt,Yt)|P]) :-
    connected(room(X,Y),room(Xt,Yt)) .

travel(room(X,Y),room(Xt,Yt),Visited,Path) :-
    connected(room(X,Y),room(Xnew,Ynew)),
    visited(room(Xnew,Ynew)),
    room(Xnew,Ynew) \== room(Xt,Yt),
    \+member(room(Xnew,Ynew),Visited),
    travel(room(Xnew,Ynew),room(Xt,Yt),[room(Xnew,Ynew)|Visited],Path) .

moveFromTo(room(X,Y),room(Xt,Yt),Path) :-
    hunter(room(X,Y)),
    path(room(X,Y),room(Xt,Yt),Path) .

```

The moveFromTo/3 predicate can be used to find a path between two rooms going only from rooms that have been visited and, therefore, have been confirmed to be safe. It can also be used to find all rooms you can move to from a given room (e.g., the agent's current position) and give a path that will connect them. Here, X and Y are the coordinates of the **From** room and Xt and Yt are the coordinates of the **To** room while Path is a list that containing all rooms that make up the path in order.

```

shootWumpus(room(X,Y),room(Xnew,Ynew),Path) :-
    room(X,Y),
    wumpus(X,Y),
    room(X2,Y2),
    hunter(room(X2,Y2)),
    room(Xnew,Ynew),
    adjacent(X,Y,Xnew,Ynew),
    stench(room(Xnew,Ynew)),
    moveFromTo(room(X2,Y2),room(Xnew,Ynew),Path) .

```

To kill the Wumpus, the agent has to know the location of the Wumpus and then travel to a room adjacent to that of the Wumpus so that it can shoot it from there. Xnew and Ynew are the coordinates of where the agent should be.

With all these predicates in the knowledgebase, there should be enough information to always infer what the best action would be. The rule bestAction/1 can be queried to get the optimal action to perform given the current information.

```
bestAction(Action):-
    wumpus(X2,Y2) ,
    shootWumpus(room(X2,Y2),room(Xhunter,Yhunter),Path) ,
    Action = [1,room(Xhunter,Yhunter),Path] .
```

```
bestAction(Action):-
    hunter(room(X,Y)) ,
    grabGold(room(X,Y)) ,
    Action = [2,room(X,Y),[room(X,Y)]] .
```

```
bestAction(Action):-
    hunter(room(X,Y)) ,
    room(Xt,Yt) ,
    not(visited(room(Xt,Yt))) ,
    moveFromTo(room(X,Y),room(Xt,Yt),Path) ,
    safe(room(Xt,Yt)) ,
    not(visited(room(Xt,Yt))) ,
    Action = [3,room(Xt,Yt),Path] .
```

```
bestAction(Action):-
    hunter(room(X,Y)) ,
    room(Xt,Yt) ,
    not(visited(room(Xt,Yt))) ,
    moveFromTo(room(X,Y),room(Xt,Yt),Path) ,
    not(visited(room(Xt,Yt))) ,
    Action = [4,room(Xt,Yt),Path] .
```

There are four different definitions for bestAction/1, each definition has a higher priority than the next. This means that the first solution reached is the highest priority action, it is the one that should be performed by the agent. Therefore, the highest priority for the agent is to kill the Wumpus, if not possible then grab gold in the room, if there is no room then move to an unvisited room, first to a known to be safe room, but if there is not any known safe room, the agent has to take a risk and move to an unsafe room that MAY contain a pit or a live Wumpus.

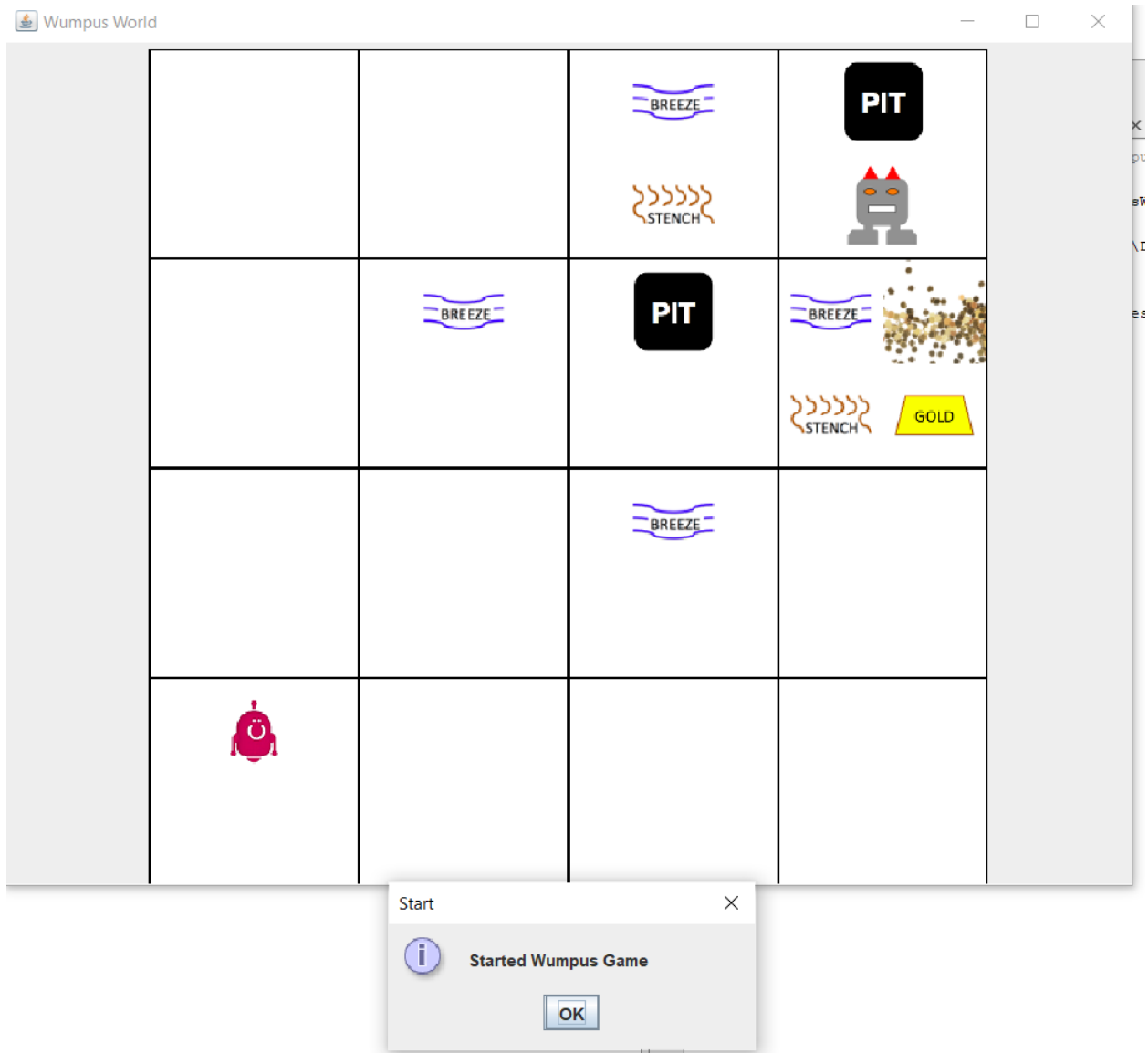
The Action variable is where the action gets returned. It contains an integer which is the action type (depending on priority), a target room for the agent to move to (which is the same if the agent just grabs the gold from the room it is located in), and a path from the agent's current location to the new one. The action is communicated to the agent through the Ask() API.

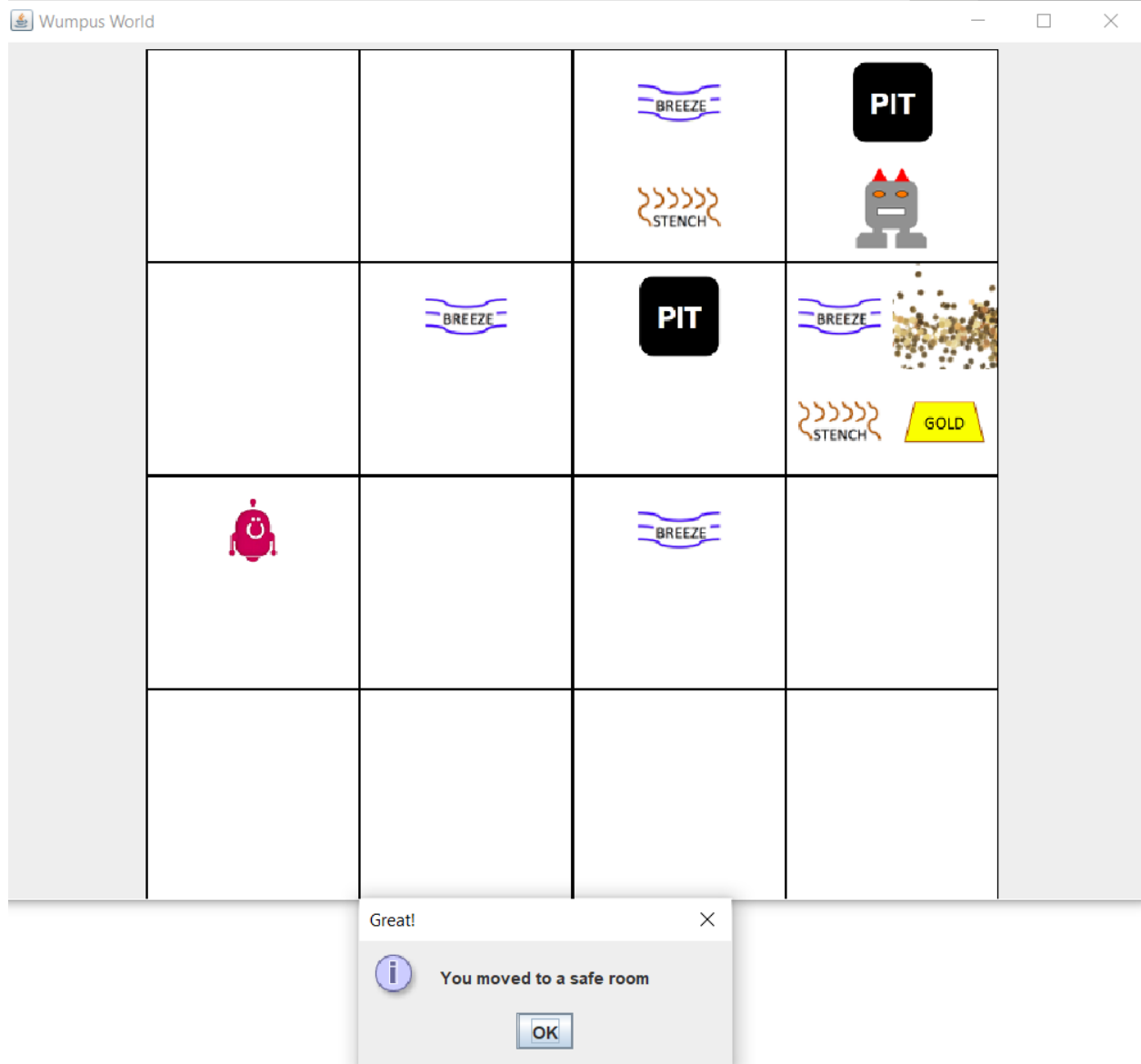
```
:- dynamic(glitter/1) .  
:- dynamic(breeze/1) .  
:- dynamic(stench/1) .  
:- dynamic(room/2) .  
:- dynamic(visited/1) .  
:- dynamic(hunter/1) .
```

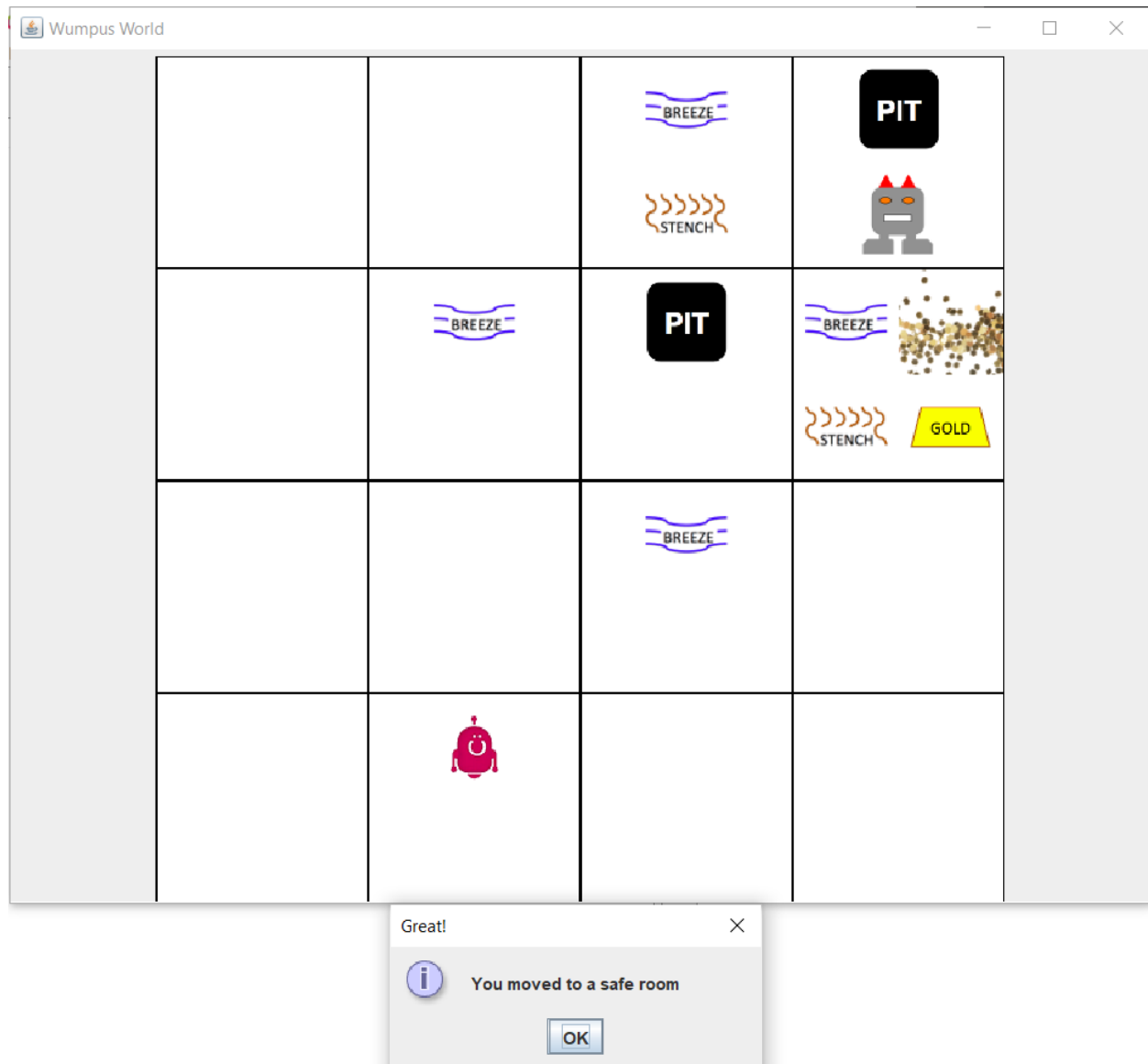
The dynamic predicates are facts that the agent can communicate to the knowledgebase using the Tell() API. The agent gets the necessary information from its actuators. This means that the agent does not know anything about the world except for what it senses on a local level, not even the dimensions of the world. When an agent travels to a room, it can know about the existence of adjacent rooms (but not their content). It can also mark a room as visited, and it can sense glitter, breeze, and/or stench in the room it is located in.

### **Successful Run (Warning, too many screenshot)**











Wumpus World

BREEZE

STENCH

PIT






BREEZE

PIT

BREEZE

STENCH



BREEZE

Great!

i

You moved to a safe room

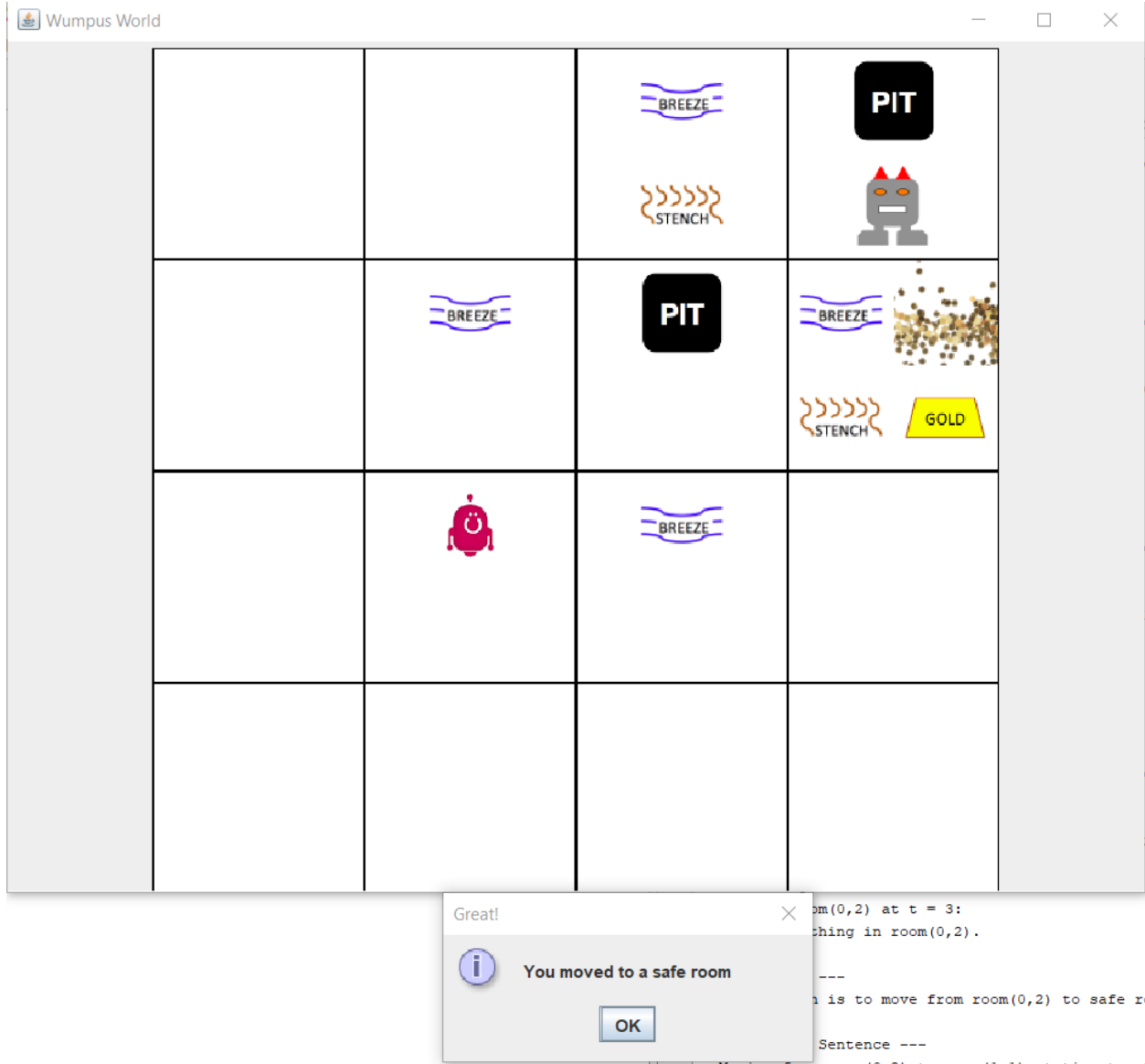
OK

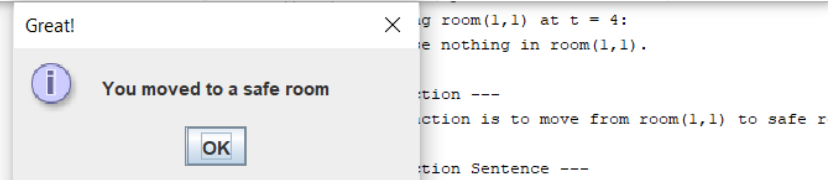
on ---

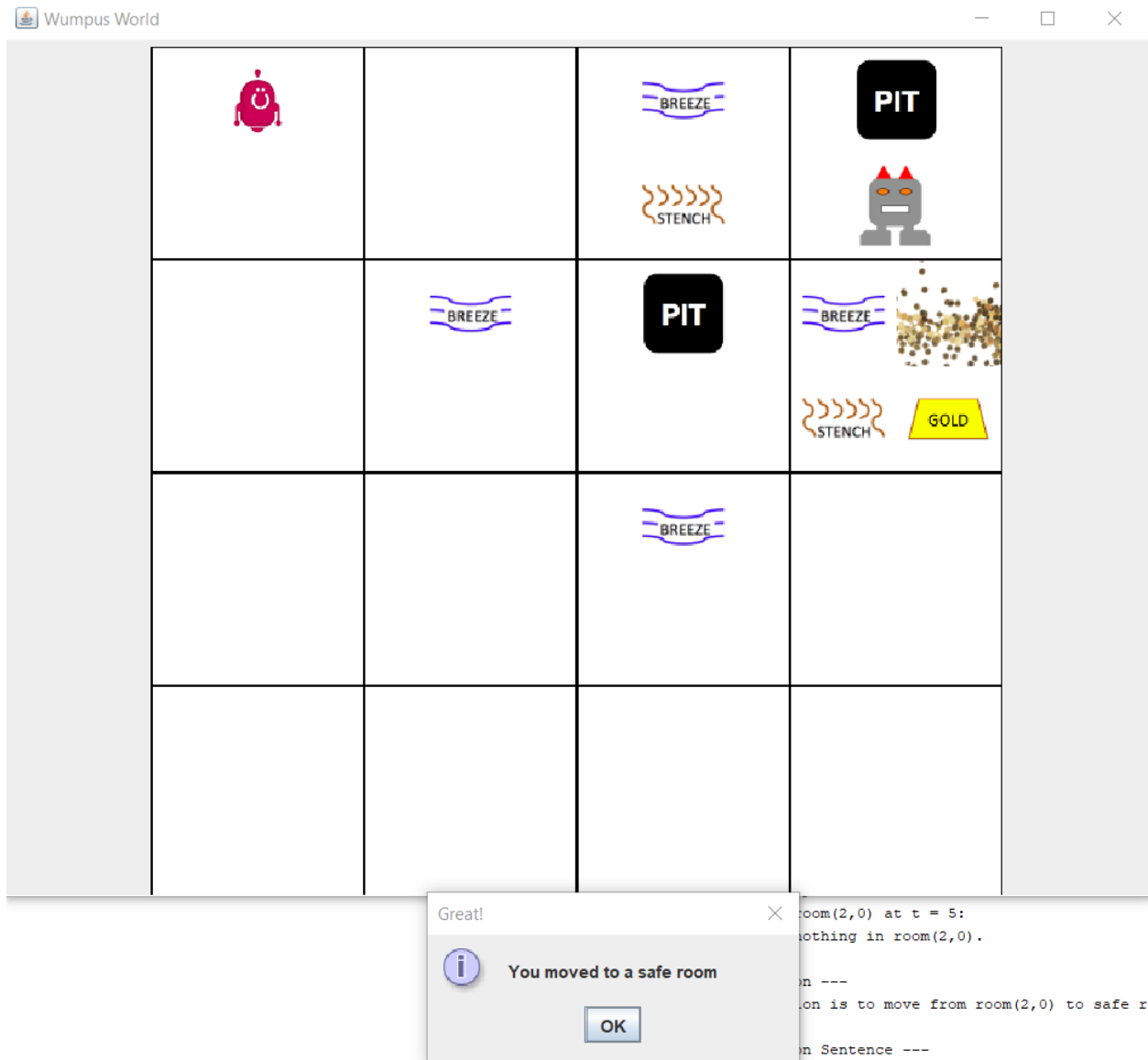
ion is to move from room(1,0) to safe r

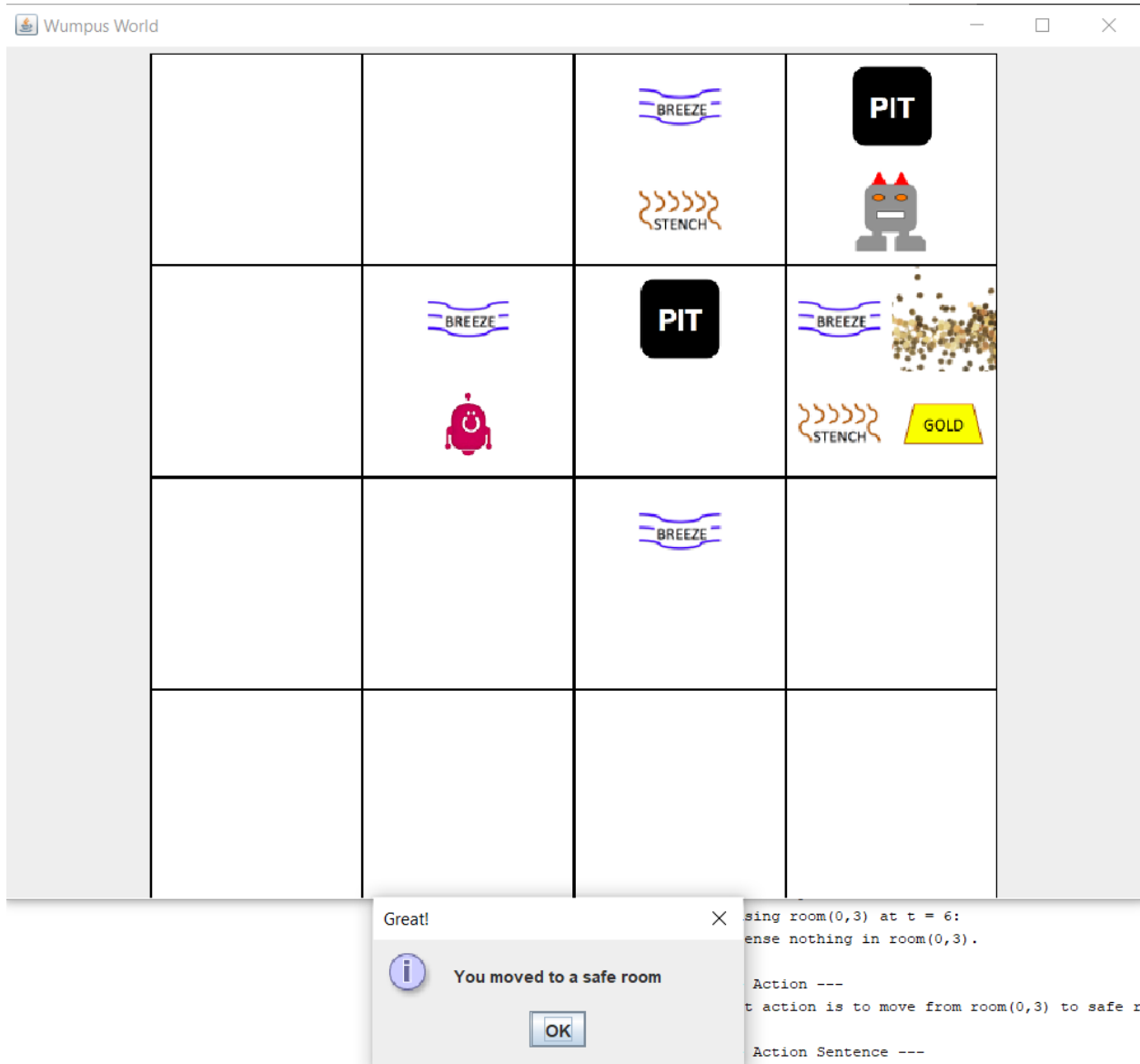
on Sentence ---

rom room(1,0) to room(0,2) at time t =

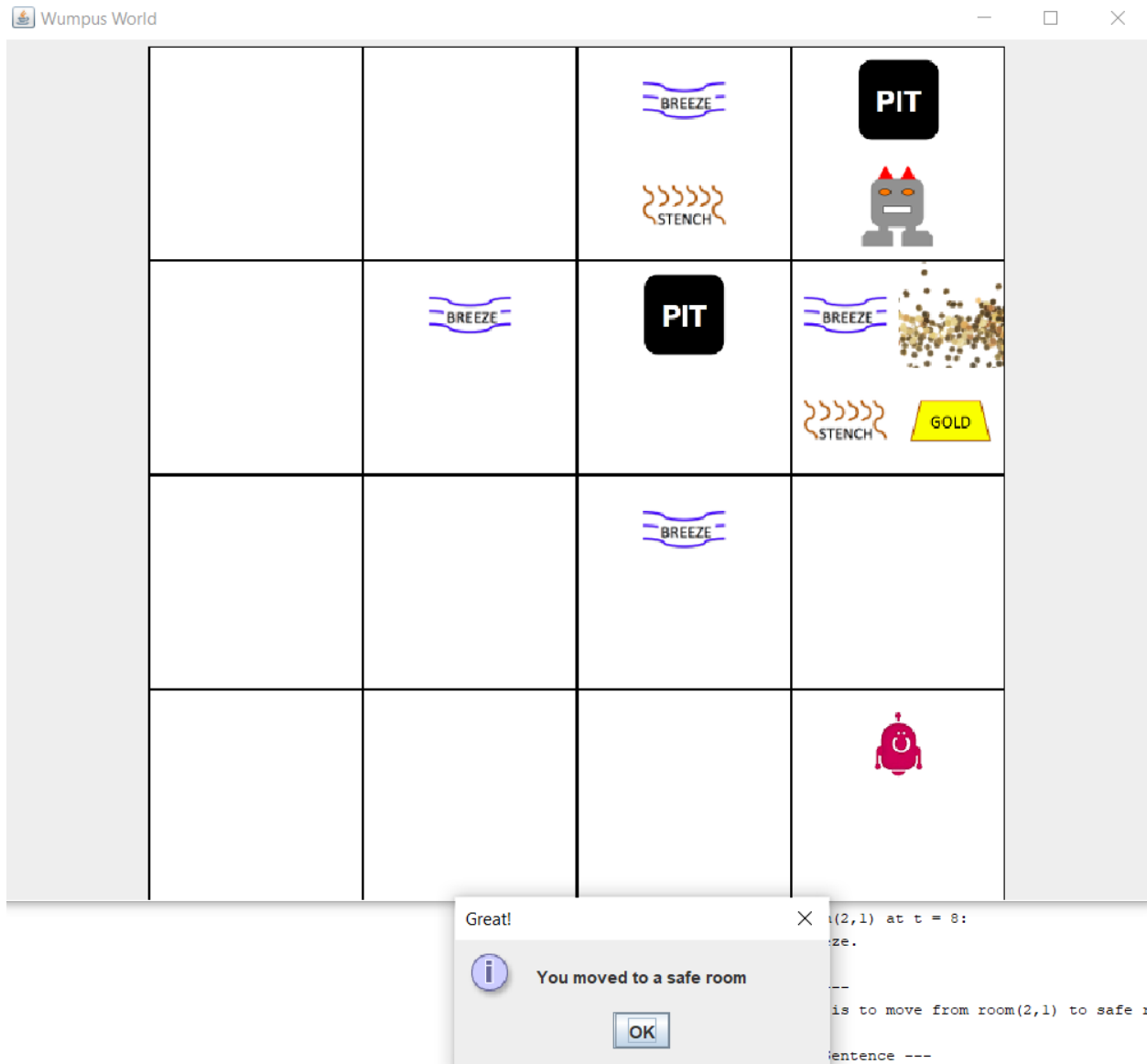


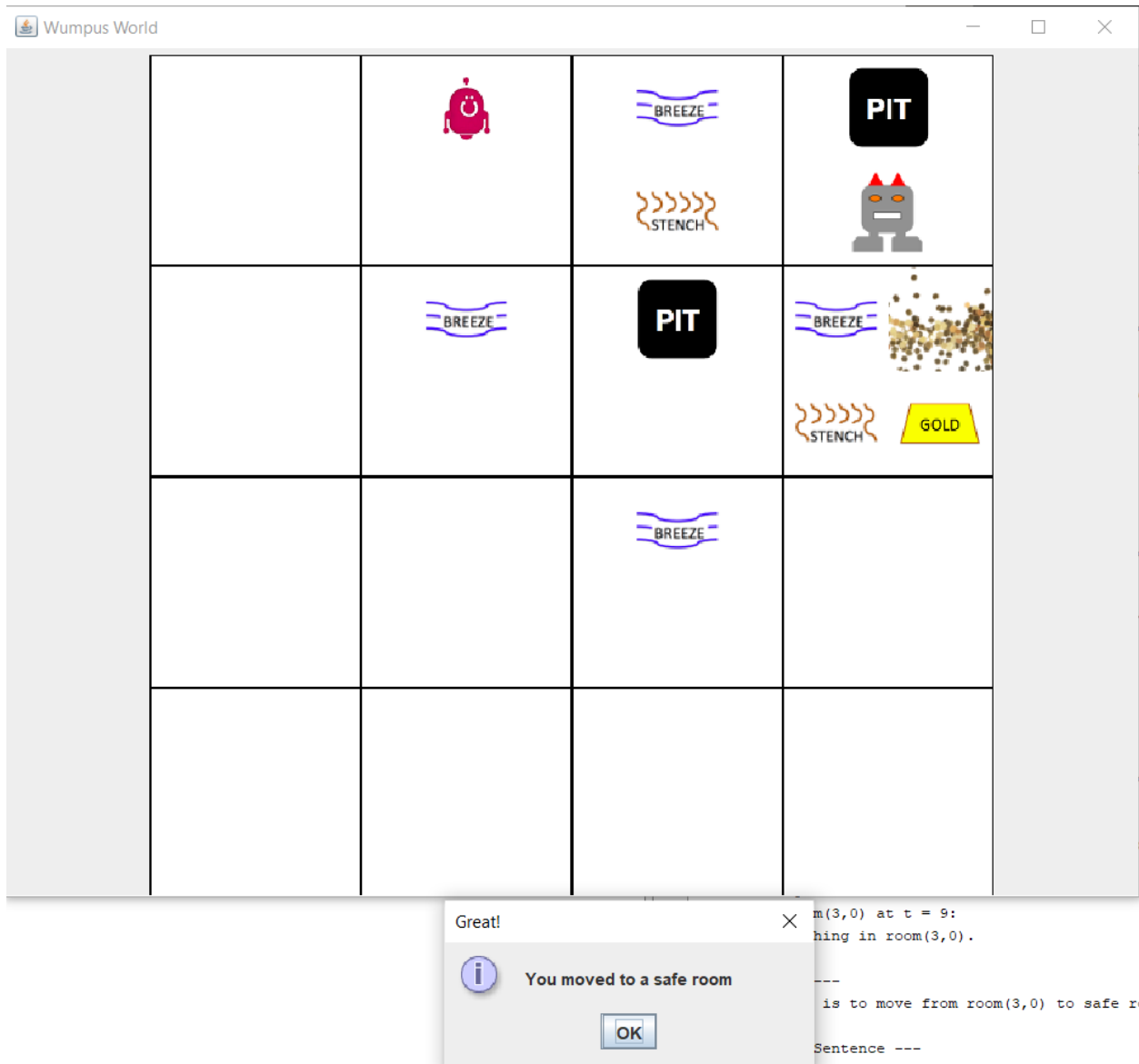


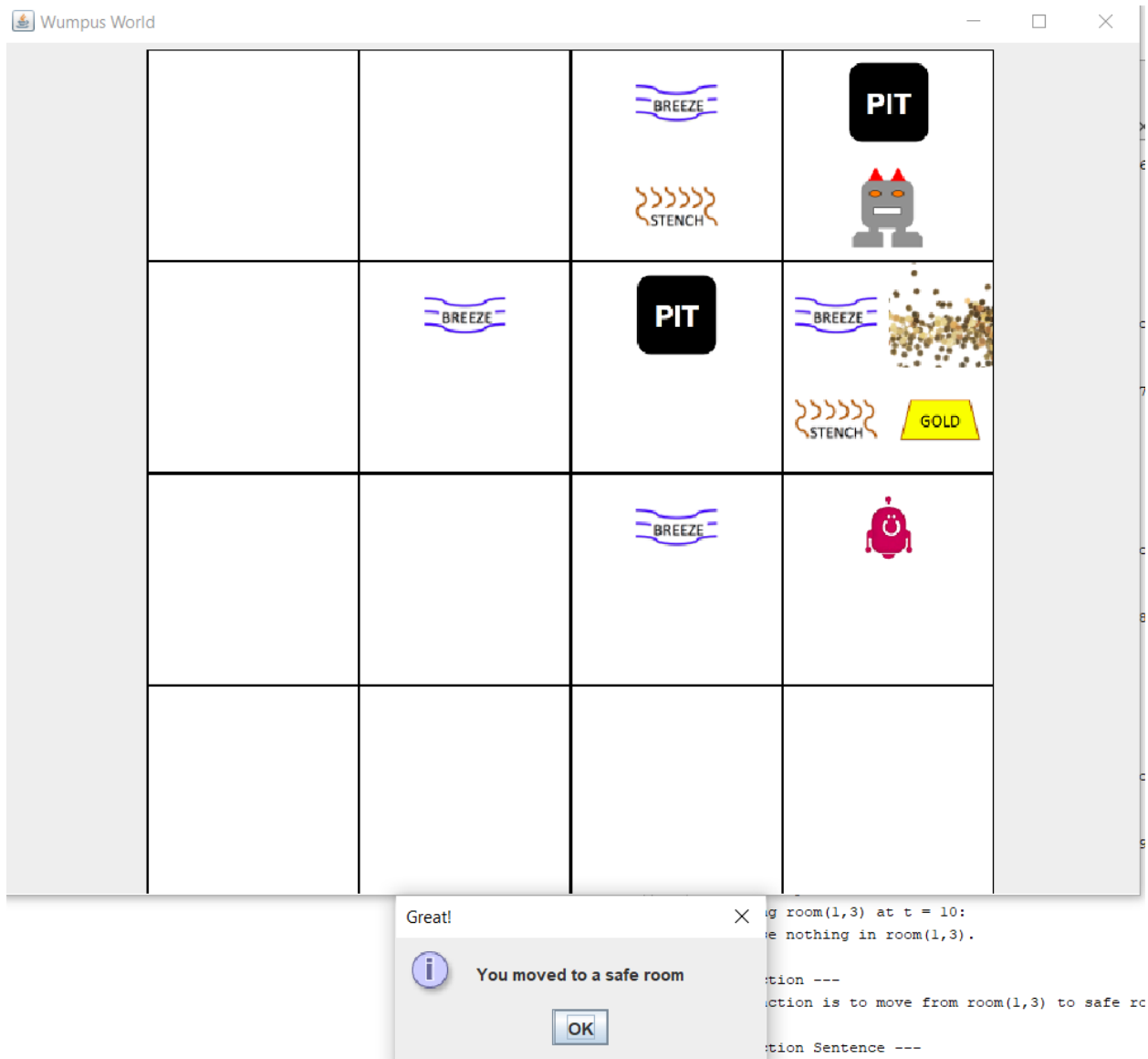


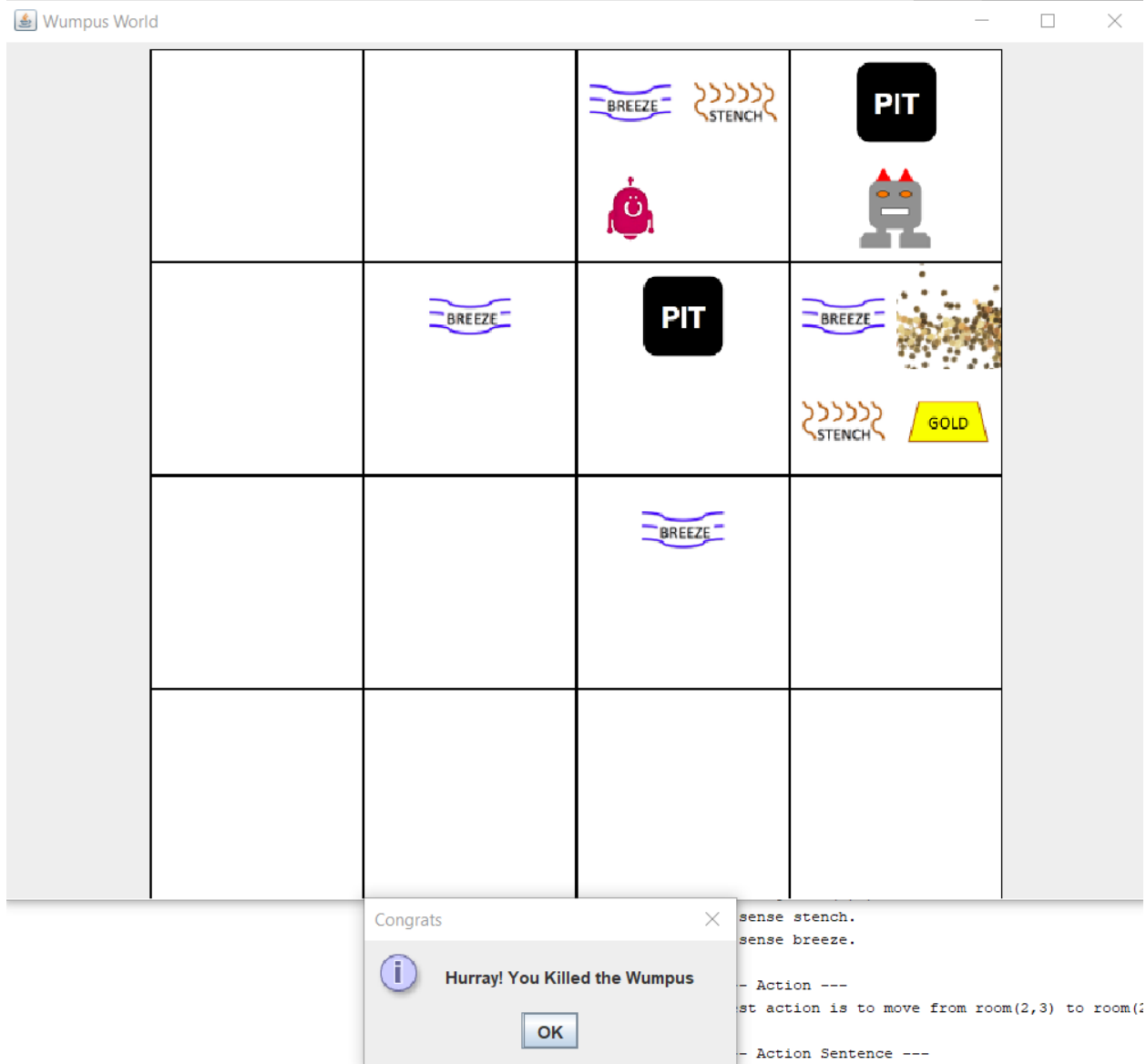












```

--- Percept Sentence ---
Sensing room(0,0) at t = 0:
I sense nothing in room(0,0).

--- Action ---
Best action is to move from room(0,0) to safe room(0,1) at time t = 0, which costs a total of 1 point(s).

--- Action Sentence ---
Moving from room(0,0) to room(0,1) at time t = 0.

--- Percept Sentence ---
Sensing room(0,1) at t = 1:
I sense nothing in room(0,1).

--- Action ---
Best action is to move from room(0,1) to safe room(1,0) at time t = 1, which costs a total of 2 point(s).

--- Action Sentence ---
Moving from room(0,1) to room(1,0) at time t = 1.

--- Percept Sentence ---
Sensing room(1,0) at t = 2:
I sense nothing in room(1,0).

--- Action ---
Best action is to move from room(1,0) to safe room(0,2) at time t = 2, which costs a total of 3 point(s).

--- Action Sentence ---
Moving from room(1,0) to room(0,2) at time t = 2.

--- Percept Sentence ---
Sensing room(0,2) at t = 3:
I sense nothing in room(0,2).

--- Action ---
Best action is to move from room(0,2) to safe room(1,1) at time t = 3, which costs a total of 2 point(s).

--- Action Sentence ---
Moving from room(0,2) to room(1,1) at time t = 3.

```

```

--- Percept Sentence ---
Sensing room(1,1) at t = 4:
I sense nothing in room(1,1).

--- Action ---
Best action is to move from room(1,1) to safe room(2,0) at time t = 4, which costs a total of 2 point(s).

--- Action Sentence ---
Moving from room(1,1) to room(2,0) at time t = 4.

--- Percept Sentence ---
Sensing room(2,0) at t = 5:
I sense nothing in room(2,0).

--- Action ---
Best action is to move from room(2,0) to safe room(0,3) at time t = 5, which costs a total of 5 point(s).

--- Action Sentence ---
Moving from room(2,0) to room(0,3) at time t = 5.

--- Percept Sentence ---
Sensing room(0,3) at t = 6:
I sense nothing in room(0,3).

--- Action ---
Best action is to move from room(0,3) to safe room(1,2) at time t = 6, which costs a total of 2 point(s).

--- Action Sentence ---
Moving from room(0,3) to room(1,2) at time t = 6.

--- Percept Sentence ---
Sensing room(1,2) at t = 7:
I sense breeze.

--- Action ---
Best action is to move from room(1,2) to safe room(2,1) at time t = 7, which costs a total of 2 point(s).

--- Action Sentence ---
Moving from room(1,2) to room(2,1) at time t = 7.

--- Percept Sentence ---
Sensing room(2,1) at t = 8:
I sense breeze.

```

```

--- Percept Sentence ---
Sensing room(2,1) at t = 8:
I sense breeze.

--- Action ---
Best action is to move from room(2,1) to safe room(3,0) at time t = 8, which costs a total of 2 point(s).

--- Action Sentence ---
Moving from room(2,1) to room(3,0) at time t = 8.

--- Percept Sentence ---
Sensing room(3,0) at t = 9:
I sense nothing in room(3,0).

--- Action ---
Best action is to move from room(3,0) to safe room(1,3) at time t = 9, which costs a total of 9 point(s).

--- Action Sentence ---
Moving from room(3,0) to room(1,3) at time t = 9.

--- Percept Sentence ---
Sensing room(1,3) at t = 10:
I sense nothing in room(1,3).

--- Action ---
Best action is to move from room(1,3) to safe room(3,1) at time t = 10, which costs a total of 6 point(s).

--- Action Sentence ---
Moving from room(1,3) to room(3,1) at time t = 10.

--- Percept Sentence ---
Sensing room(3,1) at t = 11:
I sense nothing in room(3,1).

--- Action ---
Best action is to move from room(3,1) to safe room(2,3) at time t = 11, which costs a total of 11 point(s).

--- Action Sentence ---
Moving from room(3,1) to room(2,3) at time t = 11.

--- Percept Sentence ---
Sensing room(2,3) at t = 12:
I sense stench.

```

```

--- Percept Sentence ---
Sensing room(2,3) at t = 12:
I sense stench.
I sense breeze.

--- Action ---
Best action is to move from room(2,3) to room(2,3) to kill adjacent wumpus at time t = 12, which costs a total of 10 point(s).

--- Action Sentence ---
to kill Wumpus in adjacent room at time t = 12.

BUILD SUCCESSFUL (total time: 8 minutes 44 seconds)

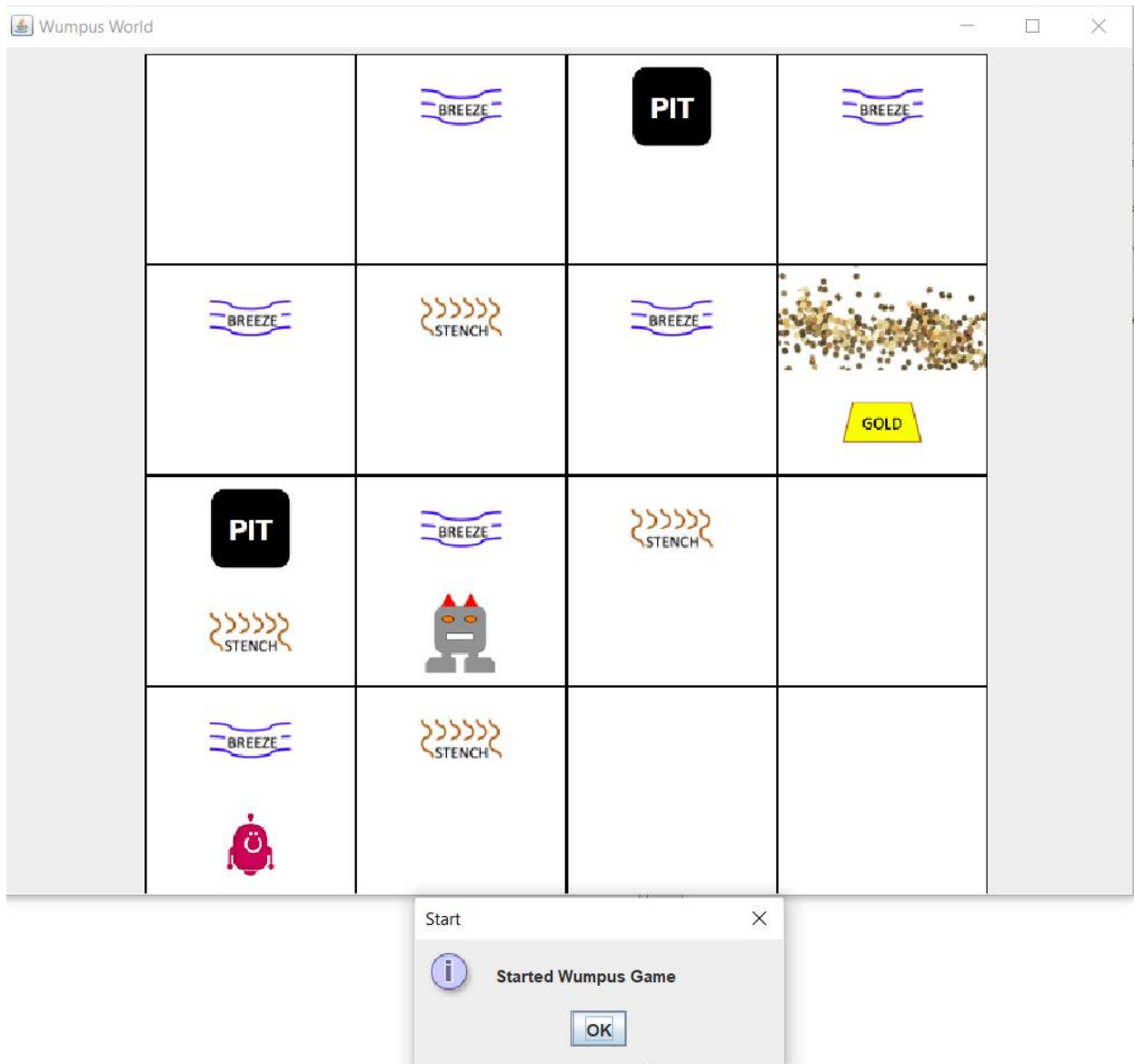
```

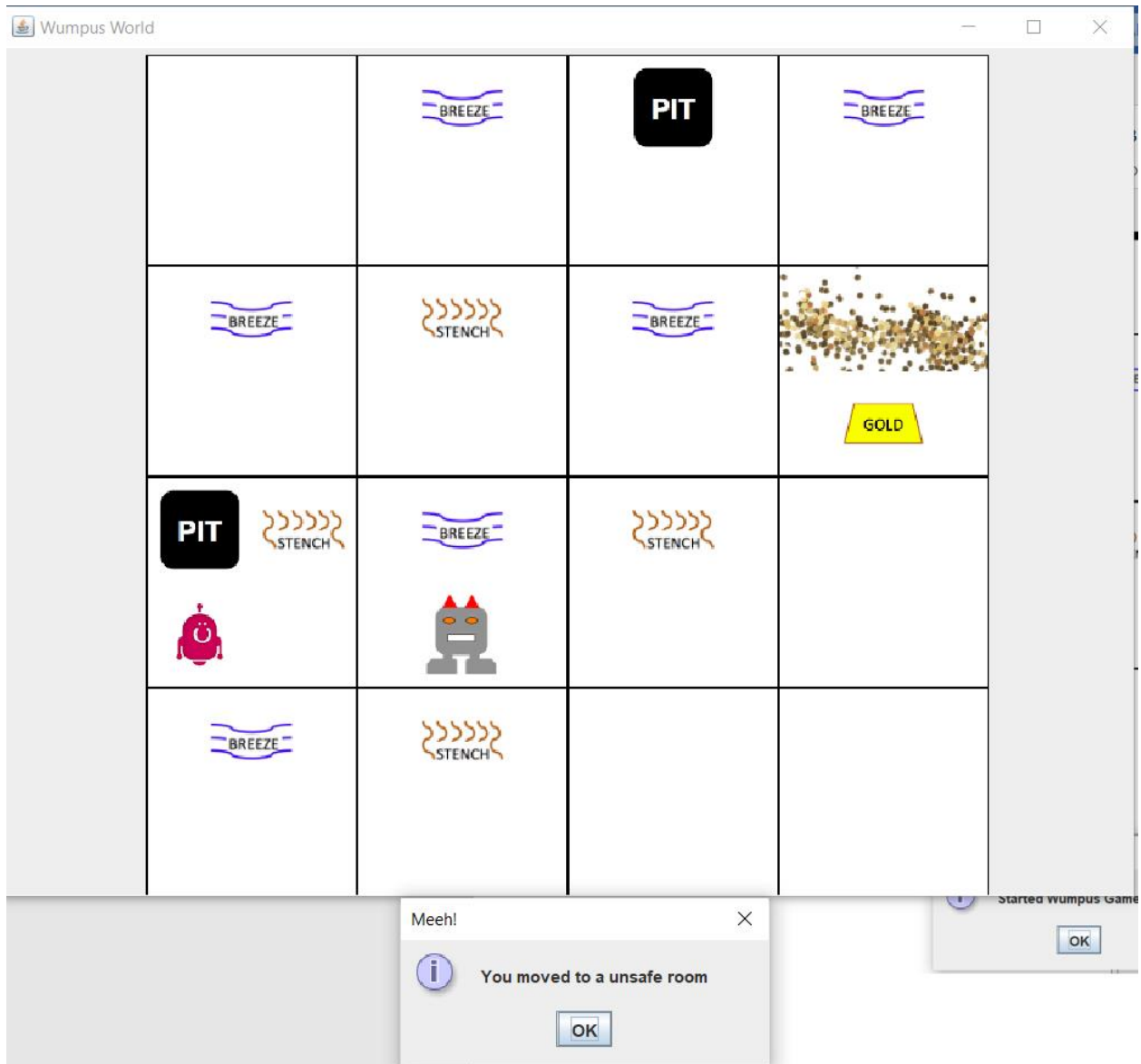
This is a typical successful case where the agent beats the game.

The information related to the APIs are printed in the console.

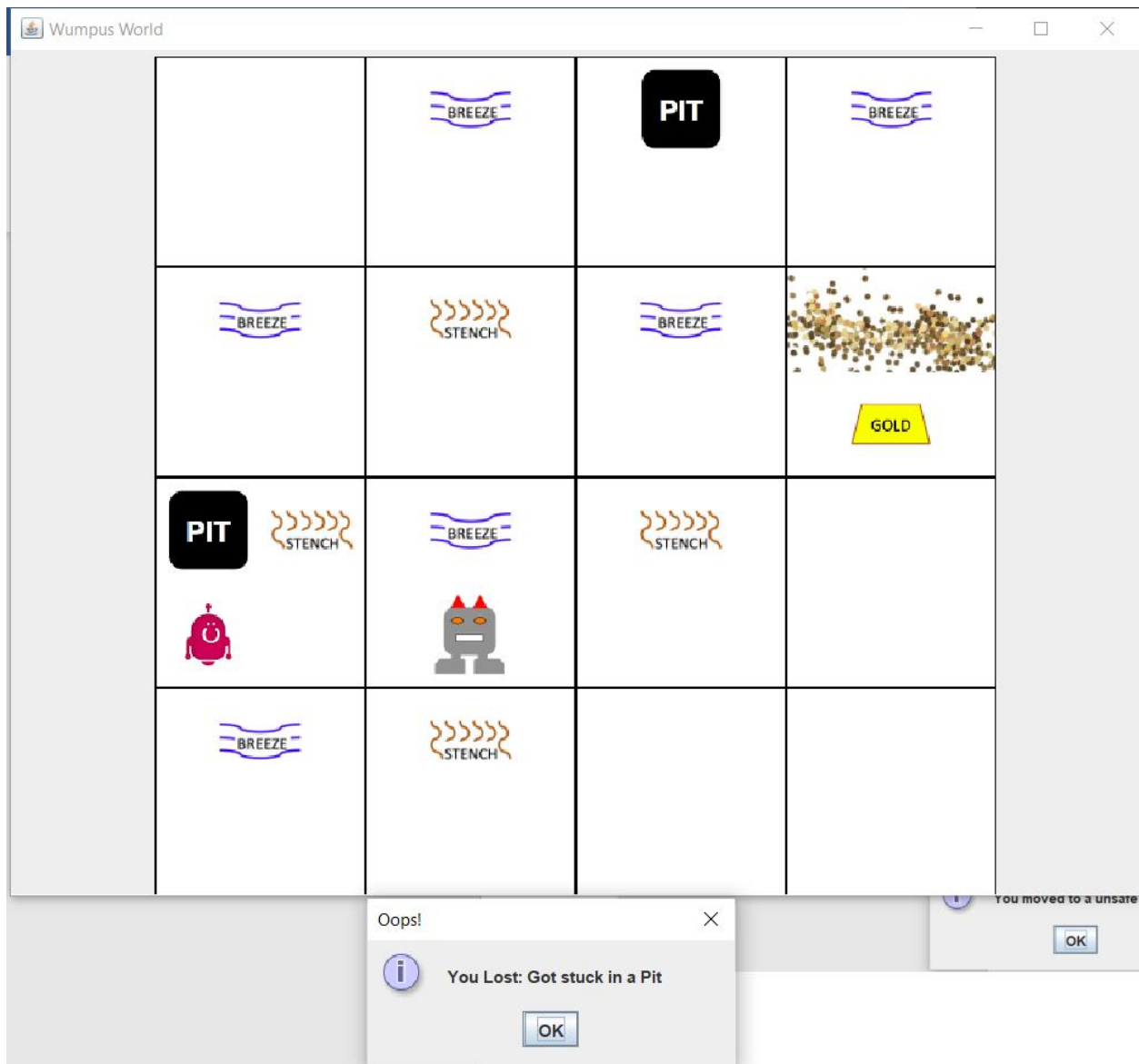
## Failed Case

Sometimes, the agent cannot solve the game. Excluding configurations that are totally unbeatable because there is no safe path leading to the Wumpus, there are cases where the agent cannot make a rational decision and, therefore, has to move to an unsafe room and risk losing the game in order to continue. The following is one such case.









This is one case where the agent sensed a breeze in room 0,0. Although there was a valid path in going right to room 1,0, it had no way to know that. It just moved to an unsafe room which happened to have a pit and just lost there.

### Weaknesses

The agent moves rationally, but there are some cases where it may never find a solution.

First, since bestAction/1 does not have any randomness built into it, the agent moves to the first unsafe room it finds when it is forced to. Meaning that if we keep running the same world configuration, the agent will always make the same moves. If those moves lead to the death of the agent, it will still repeat them.

Secondly, there are configurations of the world where the agent may come across a stench but is not able to reach and sense enough rooms to infer for certain where the Wumpus is located. As long as the agent does not know where the Wumpus is for certain, it will not attempt to shoot. In these configurations, the agent will likely keep exploring unsafe rooms until it hits a pit or a Wumpus and loses as a consequence.

## **Conclusion**

This project has been a great way to experiment with Artificial Intelligence through a logical, knowledge-based, approach. The project is missing a few features due to lack of time like a score system and a way to pick actions at random when no optimal action is found.